Master Thesis

Analysis of approximation algorithms for the traveling salesman problem in near-metric graphs

Author(s):
Krug, Sacha

Publication Date:
2011-06

Permanent Link:
https://doi.org/10.3929/ethz-a-006519144

Rights / License:
In Copyright - Non-Commercial Use Permitted
Master’s Thesis

Analysis of Approximation Algorithms for the Traveling Salesman Problem in Near-Metric Graphs

Sacha Krug
June 10th, 2011

Supervisor: Prof. Dr. Juraj Hromkovič
# Contents

1 Introduction 7

2 Preliminaries 13
   2.1 Graph Theory ........................................... 13
   2.2 Complexity Theory ...................................... 15
   2.3 Optimization ........................................... 16

3 The Path Matching Christofides Algorithm 19
   3.1 The Algorithm ........................................... 21
      3.1.1 Construction of a Minimum Path Matching ....... 22
      3.1.2 Conflict Resolution in the Minimum Path Matching ... 22
      3.1.3 Conflict Resolution in the Minimum Spanning Tree ... 26
      3.1.4 Conflict Resolution in the Modified Eulerian Cycle ... 26
      3.1.5 Completeness of the Solution ....................... 31
   3.2 An Example ............................................. 32
      3.2.1 Minimum Spanning Tree ............................. 35
      3.2.2 Minimum Path Matching ............................. 35
      3.2.3 Conflict Resolution in the Minimum Path Matching ... 37
      3.2.4 Eulerian Cycle ..................................... 37
      3.2.5 Conflict Resolution in the Minimum Spanning Tree ... 38
      3.2.6 Conflict Resolution in the Modified Eulerian Cycle ... 39

4 On the Approximation Ratio of the PMCA 41
   4.1 Minimum Spanning Tree ................................. 43
   4.2 Minimum Path Matching ................................. 44
   4.3 Conflict Resolution in the Minimum Path Matching .... 48
   4.4 Eulerian Cycle .......................................... 49
   4.5 Conflict Resolution in the Minimum Spanning Tree .... 50
   4.6 Conflict Resolution in the Modified Eulerian Cycle ... 50
5 The PMCA for the Hamiltonian Path Problem 53
5.1 The Algorithm .............................................. 54
  5.1.1 Construction of a Minimum Path Matching .......... 55
  5.1.2 Conflict Resolution in the Minimum Path Matching .. 56
  5.1.3 Construction of an Eulerian Path ...................... 58
  5.1.4 Conflict Resolution in the Minimum Spanning Tree .. 59
  5.1.5 Conflict Resolution in the Modified Eulerian Path .. 59
  5.1.6 Completeness of the Solution ......................... 60
5.2 An Example .................................................. 61
  5.2.1 Minimum Spanning Tree ................................ 62
  5.2.2 Minimum Path Matching ................................ 62
  5.2.3 Conflict Resolution in the Minimum Path Matching .. 63
  5.2.4 Eulerian Path ........................................... 64
  5.2.5 Conflict Resolution in the Minimum Spanning Tree .. 64
  5.2.6 Conflict Resolution in the Modified Eulerian Path .. 65

6 On the Approximation Ratio of the PMCA-HPP_\ell 67
6.1 The PMCA-HPP_0 ............................................ 67
  6.1.1 Minimum Spanning Tree ................................ 68
  6.1.2 Minimum Path Matching ................................. 69
  6.1.3 Conflict Resolution in the Minimum Path Matching .. 72
  6.1.4 Eulerian Path ............................................ 73
  6.1.5 Conflict Resolution in the Minimum Spanning Tree .. 74
  6.1.6 Conflict Resolution in the Modified Eulerian Path .. 74
6.2 The PMCA-HPP_1 ............................................ 76
  6.2.1 Minimum Spanning Tree ................................ 76
  6.2.2 Minimum Path Matching ................................. 77
  6.2.3 Conflict Resolution in the Minimum Path Matching .. 78
  6.2.4 Eulerian Path ............................................ 79
  6.2.5 Conflict Resolution in the Minimum Spanning Tree .. 79
  6.2.6 Conflict Resolution in the Modified Eulerian Path .. 80
6.3 The PMCA-HPP_2 ............................................ 82
  6.3.1 Minimum Spanning Tree ................................ 85
  6.3.2 Minimum Path Matching ................................. 86
  6.3.3 Conflict Resolution in the Minimum Path Matching .. 89
  6.3.4 Eulerian Path ............................................ 90
  6.3.5 Conflict Resolution in the Minimum Spanning Tree .. 91
  6.3.6 Conflict Resolution in the Modified Eulerian Path .. 92


7 Additional Results 95
  7.1 Approximation of the Metric HPP_2 96
  7.2 1.4-Approximation of the ∆-LM-TSP 100
  7.3 4/3-Approximation of the ∆-LM-TSP 109

8 Conclusion 113

Bibliography 115

Contents
Chapter 1

Introduction

Certain problems in computer science are harder to solve than others. More precisely, the runtime of any deterministic, exact algorithm for such a problem may be exponential in the input length. Solving such problems exactly can take millions of years even on the fastest computers.

This is clearly not feasible. Instead, one possible approach is to search for a solution that is as close to the optimum as possible. These problems are therefore called optimization problems, and the algorithms that compute approximate solutions are called approximation algorithms. Such approximation algorithms need significantly less time than exact algorithms, e.g. only a few seconds on an ordinary personal computer. An optimization problem can be categorized according to how close the best known approximation algorithm for it comes to the optimal solution in the worst case.

One of the hardest optimization problems is the traveling salesman problem (TSP).\textsuperscript{1} Initially, it was stated as follows. A traveling salesman needs to visit a certain number of cities. He knows the distances between all these cities, and he wants to find a tour through all these cities that is as short as possible and that ends in the city where he started. More formally, we can formulate the problem as follows. Given a complete graph with edge costs, find a cycle that visits every vertex exactly once and has minimum overall edge cost. The TSP has many applications including public transportation systems, scheduling, routing, and logistics [Coo07, LLRS85].

The metric TSP (or $\Delta$-TSP) is the TSP restricted to graphs satisfying the triangle inequality

$$c({v, w}) \leq c({v, u}) + c({u, w}),$$

\textsuperscript{1}It is NP-hard, i.e., every deterministic, exact algorithm needs exponential time to solve it in the worst case (if $P \neq NP$).
for any three vertices \( u, v, w \). The best known algorithm for the \( \Delta \)-TSP is Christofides’ algorithm [Chr76], which guarantees a solution that has at most 3/2 times the cost of an optimal solution.

The triangle inequality can be generalized in order to refine the categorization of the TSP. The \( \beta \)-metric TSP (or \( \Delta_{\beta} \)-TSP) is the TSP restricted to graphs satisfying the \( \beta \)-triangle inequality

\[
c(\{v, w\}) \leq \beta \cdot (c(\{v, u\}) + c(\{u, w\}))
\]

for any three vertices \( u, v, w \) [BCS94]. It is easy to see that \( \beta \geq 1/2 \), and the case \( \beta = 1/2 \) is the trivial case where all edges in the graph have the same cost.

The first goal of this thesis is to analyze an approximation algorithm for the \( \Delta_{\beta} \)-TSP, for \( \beta \geq 1 \). The second goal is to analyze a variant of this algorithm for the Hamiltonian path problem (HPP). The Hamiltonian path problem is the following problem. Given a complete graph with edge costs, find a path between two vertices that visits every vertex exactly once and has minimum overall edge cost. Informally, the problem can be described as that of the traveling salesman above, with the only difference that he needs to end up in a different city than the one where he started. There exist three variants of this problem: with no fixed endpoints, with one fixed endpoint, and with two fixed endpoints.\(^2\)

**Previous Work**

The cube of a connected graph \( G = (V, E) \) is the graph \( G^3 = (V, E') \) with edges between all vertices that are connected by a path with at most three edges in \( G \). Sekanina [Sek60] showed that the cube of any connected graph is Hamiltonian, i.e., in particular that, if \( T \) is a minimum spanning tree of a graph, then \( T^3 \) is Hamiltonian. Using this result, Andreae and Bandelt [AB95] constructed an approximation algorithm for the \( \Delta_{\beta} \)-TSP with an approximation ratio bounded from above by \( \frac{3}{2} \beta^2 + \frac{1}{2} \beta \). They also gave an example that proves a lower bound of \( \beta^2 + \beta - \varepsilon \) and conjectured that “a more careful analysis of the \( T^3 \)-algorithm [...] would yield” the tightness of this lower bound. Andreae [And01] verified this conjecture by showing an upper bound of \( \beta^2 + \beta \) on the approximation ratio of a refined version of the algorithm.

Christofides’ algorithm [Chr76], whose approximation ratio is bounded from above by 3/2, is a well-known approximation algorithm for the metric

\(^2\)The fixed endpoints are part of the problem description.
TSP and is taught in many undergraduate introductory courses. Despite its simplicity, no better approximation algorithm has yet been found for the metric TSP. Böckenhauer et al. [BHK+00] generalized the upper bound to \( \frac{3^\beta}{3^\beta - 2^{\beta + 1}} \), for \( \beta \leq 1 \). They also adapted a cycle-cover algorithm for the asymmetric TSP due to Frieze, Galbiati and Maffioli [FGM82] and obtained a better approximation ratio of \( \frac{\beta - 2}{3(\beta - 1)} \), for \( \beta < 2/3 \). Sprock [Spr08] proved a lower bound of \((2^\beta + \beta + 5)/6 - \varepsilon\) for the Christofides algorithm, for \( \beta \leq 1 \).

Bender and Chekuri [BC00] came up with an approximation algorithm for the near-metric TSP that has an approximation ratio of \( 4\beta \). Their algorithm relies on a result by Fleischner [Fle74] that the square of a 2-connected graph is Hamiltonian and on a constructive proof of this result due to Lau [Lau80].

Finally, Böckenhauer et al. [BHK+02] devised the so-called path matching Christofides algorithm (PMCA) with an approximation ratio bounded from above by \( \frac{3^\beta}{2^\beta} \), thus outperforming both Sekanina’s and Bender and Chekuri’s algorithm, for \( 1 \leq \beta \leq 2 \). Especially nice about this algorithm is that it is a very natural expansion of the Christofides algorithm for near-metric graphs.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Range</th>
<th>Upper bound</th>
<th>Lower bound</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cycle-cover</td>
<td>( \frac{1}{2} \leq \beta \leq \frac{2}{3} )</td>
<td>( \frac{\beta - 2}{3(\beta - 1)} ) [BHK+00]</td>
<td>–</td>
</tr>
<tr>
<td>Christofides</td>
<td>( \frac{2}{3} \leq \beta \leq 1 )</td>
<td>( \frac{\beta - 2}{3(\beta - 1)} ) [BHK+00]</td>
<td>( \frac{2^{\beta^2 + \beta + 5}}{6} - \varepsilon ) [Spr08]</td>
</tr>
<tr>
<td>PMCA</td>
<td>( 1 \leq \beta \leq 2 )</td>
<td>( \frac{3^\beta}{2^\beta} ) [BHK+02]</td>
<td>–</td>
</tr>
<tr>
<td>HCT(^3) refined</td>
<td>( 2 \leq \beta \leq 3 )</td>
<td>( \beta^2 + \beta ) [And01]</td>
<td>( \beta^2 + \beta - \varepsilon ) [AB95]</td>
</tr>
<tr>
<td>Bender-Chekuri</td>
<td>( \beta \geq 3 )</td>
<td>( 4\beta ) [BC00]</td>
<td>–</td>
</tr>
</tbody>
</table>

Table 1.1: The best currently known \( \Delta_\beta \)-TSP approximation algorithms.

Table 1.1 summarizes these results. The “–” in the lower bound column means no lower bound has been proved specifically for this algorithm. The general lower bounds on the approximability of the \( \Delta_\beta \)-TSP and the \( \Delta \)-TSP of \( \frac{3803 + 10\beta}{3804 + 8\beta} - \varepsilon \) [BS00] and \( \frac{220}{219} - \varepsilon \) [PV06], respectively, of course always apply, independent of the algorithm.
The Hamiltonian path problem has not been studied as extensively as the TSP. However, many interesting algorithmic ideas from the TSP can be adapted for the Hamiltonian path problem, see e.g. [FHPS06].

Contribution of this Thesis

In the first part of this thesis, we analyze the path matching Christofides algorithm (PMCA) due to Böckenhauer et al. [BHK+02]. The authors established an upper bound of $\frac{3}{2} \beta^2$ on the approximation ratio of the PMCA. In this thesis, we first correct an implementation error by the authors and then show that the upper bound is tight, i.e., that there are worst-case instances on which the PMCA cannot achieve an approximation ratio of $\frac{3}{2} \beta^2 - \varepsilon$, for an arbitrarily small $\varepsilon > 0$.

In the second part, we analyze the PMCA variants for the Hamiltonian path problem due to Forlizzi et al. [FHPS06]. The algorithms are called PMCA-HPP$^l$, where $l \in \{0, 1, 2\}$ is the number of prespecified endpoints. The authors proved an upper bound on the approximation ratio of $\frac{3}{2} \beta^2$, for $l \in \{0, 1\}$, and of $\frac{5}{3} \beta^2$, for $l = 2$. We prove the tightness of these upper bounds by providing worst-case instances on which the algorithms cannot achieve an approximation ratio of $\frac{3}{2} \beta^2 - \varepsilon$ for the first two problems and of $\frac{5}{3} \beta^2 - \varepsilon$ for the latter problem, for an arbitrarily small $\varepsilon > 0$.

In the last part, we analyze an approximation algorithm for the metric Hamiltonian path problem due to Forlizzi et al. [FHPS06]. We show that the upper bound of 3 on the approximation ratio is tight. Then we analyze two algorithms for the metric TSP reoptimization due to Böckenhauer et al. [BFH+07] and Berg and Hempel [BH09], respectively. For the case that the edge weight is increased, we establish lower bounds of $4/3 - \varepsilon$ on the approximation ratio of both algorithms, which is tight in the latter case.

The implications of these results are twofold. On the one hand, we know now that most of these algorithms can indeed return a result that is as bad as possible, i.e., the upper bounds cannot be improved. So if we want algorithms that provide better upper bounds, we have to come up with new ones. On the other hand, the structure of the worst-case examples may provide insights into why the algorithms perform badly and may help devise better algorithms.

Outline

This thesis is organized as follows. In Chapter 2, we introduce all important definitions and concepts. We assume the reader to be familiar with the
material, therefore this chapter is concise.

Chapter 3 is devoted to the path matching Christofides algorithm (PMCA). After presenting the algorithm, we correct an implementation error in the original paper [BHK+02] and give an example to get the reader familiar with the algorithm.

In Chapter 4, we show that the upper bound of $\frac{3}{2}\beta^2$ on the approximation ratio of the PMCA established in [BHK+02] is tight.

Chapters 5 and 6 are similar to the previous two. In Chapter 5, we present a combination devised in [FHPS06] of Hoogeveen’s [Hoo91] ideas and the PMCA resulting in an approximation algorithm for the Hamiltonian path problem with $l \in \{0, 1, 2\}$ fixed endpoints (HPP$_l$).

In Chapter 6, we prove that the upper bounds on the approximation ratio of the PMCA variants for HPP$_0$, HPP$_1$, and HPP$_2$ of $\frac{3}{2}\beta^2$ for the first two and $\frac{5}{3}\beta^2$ for the latter established in [FHPS06] are tight.

In Chapter 7, we improve known lower bounds for some selected HPP$_2$ and TSP approximation algorithms.

Chapter 8 summarizes the results of this thesis and discusses open problems.

Acknowledgments

First and foremost, I want to thank Hans-Joachim Böckenhauer for his constant support throughout the creation of this thesis. Our weekly meetings were immensely valuable to show me where I was standing. He always took his time to answer my questions in great detail, for which I am grateful as it increased my already strong interest in research even more. Also, I want to thank him for proofreading every chapter; his feedback significantly improved the quality of this thesis.

Also, I want to thank Professor Juraj Hromkovič for giving me the possibility to write this thesis in his group. He was very helpful in finding an interesting topic that suited my knowledge, and he saw the big picture of this thesis even before I had started writing it. Later, at our midterm discussion, he recommended to pursue the current topic further; this turned out to be just the right decision, as I could tackle the problem completely.

Last but not least, I want to thank my mother for her love and support before, during, and after the writing of this thesis.
Chapter 1. Introduction
Chapter 2

Preliminaries

We assume the reader to be familiar with the following topics and only give a brief overview of the most important terms and concepts. The structure of this chapter is for the most part adopted from [Spr08]. For more detailed introductions to graph theory, complexity theory, and optimization, we refer the interested reader to [Die10, Wes01], [GJ79, Hro10, Sip06], and [ACG+99, Hro04], respectively.

2.1 Graph Theory

We start with some basic definitions and notations of graph theory.

Definition 2.1. A graph $G = (V, E)$ is a pair of vertices $V$ and edges $E \subseteq \binom{V}{2}$, where $\binom{V}{2} := \{\{u, v\} \mid u, v \in V, u \neq v\}$. Sometimes we also write $V(G)$ and $E(G)$ to make it clear what graph we are referring to.

A vertex $v$ is adjacent to an edge $e$ (and vice versa) and the neighbor of a vertex $w$ in a graph $G$ if $e = \{v, w\} \in E(G)$. The degree of a vertex is the number of edges adjacent to it. We call a vertex $v$ even (odd) if it has even (odd) degree.

For an edge $e = \{v, w\}$, we use the notation $e - v$ to refer to the vertex $w$.

A complete graph is a graph with $E = \binom{V}{2}$. We denote the complete graph with $n$ vertices $K_n$.

A weighted graph is a graph $G = (V, E, c)$, where $c : E \to \mathbb{Q}^+$ is called edge cost function. We write $c(v, w)$ instead of $c(\{v, w\})$ and $c(G)$ instead of $\sum_{e \in E(G)} c(E)$. We will use the terms cost and length interchangeably.

Definition 2.2. A weighted graph $G = (V, E, c)$ is called $\beta$-metric if any triangle in it satisfies the $\beta$-triangle inequality, i.e.,

$$c(v, w) \leq \beta \cdot (c(v, u) + c(u, w)), \quad \forall u, v, w \in V.$$
Chapter 2. Preliminaries

If $\beta < 1$, the graph is super-metric. If $\beta = 1$, the graph is metric. If $\beta > 1$, the graph is near-metric.

Definition 2.3. A matching in a graph $G$ is a set of pairwise vertex-disjoint edges, i.e., a set of edges $M \subseteq E(G)$ such that, for any two edges $e, e'$ in $M$, we have $e \cap e' = \emptyset$. For an edge $e = \{v, w\}$ in a matching, we say that the vertices $v$ and $w$ are matched.

A perfect matching in a graph $G$ with even $|V(G)|$ is a matching of size $|V(G)|/2$, i.e., every vertex is matched with some other vertex.

Definition 2.4. A path $P$ in a graph $G$ is an ordered list $P = (v_1, v_2, \ldots, v_k)$ of vertices such that $\{v_i, v_{i+1}\} \in E(G)$, for $1 \leq i \leq k - 1$.

The length of a path is the number of edges in it if the graph is unweighted and the sum of the edge lengths otherwise. The distance between two vertices $u$ and $v$ is the length of the shortest path from $u$ to $v$.

A vertex $u$ is an endpoint of a path $(v, \ldots, w)$ if $u = v$ or $u = w$. A vertex $u$ is internal to a path $p$ if $p$ contains $u$, but $u$ is not an endpoint of $p$.

For $k \geq 2$ paths $p_1, p_2, \ldots, p_k$ such that path $p_i$ shares one endpoint with path $p_{i-1}$ and the other endpoint with path $p_{i+1}$, for $2 \leq i \leq k - 1$, we denote the concatenation of these paths by $(p_1, p_2, \ldots, p_k)$. More formally, let $p_i = (v_{i,1}, \ldots, v_{i,l_i})$, for $1 \leq i \leq k$. Then,

$$(p_1, p_2, \ldots, p_k) := (v_{1,1}, v_{1,2}, \ldots, v_{1,l_1} = v_{2,1}, v_{2,2}, \ldots, v_{k-1,l_{k-1}} = v_{k,1}, \ldots, v_{k,l_k}).$$

A Hamiltonian path is a path that visits every vertex exactly once. An Eulerian path is a path that visits every edge exactly once.

Definition 2.5. A cycle is a path where the first and the last vertex coincide.

A Hamiltonian cycle is a cycle that visits every vertex exactly once. An Eulerian cycle is a cycle that visits every edge exactly once.

Definition 2.6. A graph is connected if there exists a path between any two vertices.

A graph is a tree if it is connected and contains no cycles. A graph is a forest if it contains no cycles. A component (or connected component) of a (not necessarily connected) graph $G$ is a maximal connected subgraph.

For any graph $G = (V, E)$, a tree $T = (V, E')$ with $E' \subseteq E$ is called a spanning tree of $G$. A spanning tree $T$ is called a minimum spanning tree (MST) of a weighted graph $G$ if there is no spanning tree $T'$ of $G$ with $c(T') < c(T)$.

1We simply use the term triangle inequality instead of 1-triangle inequality in this case.
2.2 Complexity Theory

This section contains a very brief overview of Turing machines and the complexity classes \( P \) and \( NP \).

For two functions \( f, g : \mathbb{N} \rightarrow \mathbb{R}_0^+ \), we write

\[
f \in O(g) \text{ if there exist values } c, n_0 \in \mathbb{N} \text{ such that } f(n) \leq c \cdot g(n), \text{ for all } n \geq n_0,
\]

\[
f \in \Omega(g) \text{ if } g \in O(f),
\]

\[
f \in \Theta(g) \text{ if } f \in O(g) \text{ and } g \in O(f).
\]

We do not formally introduce the concept of a Turing machine. It was first defined in [Tur36] and is nicely explained in [Hro10]. Intuitively, it consists of three parts:

- a finite memory containing the program,
- an infinitely long tape working as both input and work tape, and
- a read/write head that can move in both directions on the tape.

It can thus read an input \( w \) on the tape, process it and output whether \( w \) is in some predefined language \( L \). The Church-Turing thesis states that every intuitively computable function is computable on a Turing machine.

For its computation, the Turing machine needs a certain number of steps. We talk about an \( f(n) \)-time Turing machine if it needs at most \( f(n) \) computation steps for every input of length at most \( n \).

**Definition 2.7.** For a function \( f : \mathbb{N} \rightarrow \mathbb{R}_0^+ \), the time complexity class \( \text{TIME}(f(n)) \) is the set of all languages decidable by an \( O(f(n)) \)-time Turing machine.

A nondeterministic Turing machine (NTM) is a Turing machine that can choose between several possible next steps in some states during its computation. We talk about an \( f(n) \)-time NTM if the shortest accepting computation for every input of length at most \( n \) needs at most \( f(n) \) steps.

**Definition 2.8.** For a function \( f : \mathbb{N} \rightarrow \mathbb{R}_0^+ \), the time complexity class \( \text{NTIME}(f(n)) \) is the set of all languages decidable by an \( O(f(n)) \)-time NTM.

Let us now introduce the two most important complexity classes.

**Definition 2.9.** \( P = \bigcup_k \text{TIME}(n^k) \) is the class of languages decidable in deterministic polynomial time.

**Definition 2.10.** \( NP = \bigcup_k \text{NTIME}(n^k) \) is the class of languages decidable in nondeterministic polynomial time.
2.3 Optimization

We now formally define an optimization problem, the traveling salesman problem, and the approximation ratio.

Definition 2.11. An optimization problem is a 6-tuple $U = (\Sigma_I, \Sigma_O, L, M, c, \text{goal})$, where

- $\Sigma_I$ is the input alphabet,
- $\Sigma_O$ is the output alphabet,
- $L \subseteq \Sigma_I^*$ is the language of valid inputs,
- $M$ is a function from $L$ to $\mathcal{P}(\Sigma_O^*)$ ($M(x)$ is the set of feasible solutions for $x \in L$),
- $c$ is a cost function from $(M(x), x), x \in L$, to $\mathbb{R}^+$, and
- $\text{goal} \in \{\text{Minimum}, \text{Maximum}\}$ is the optimization goal.

We do not explicitly state the encodings of the problem instances and solutions in the input and output alphabets, respectively—the only restriction is that we exclude unary encodings. Instead, we state an optimization problem using the parameters input $L$, constraints $M$, costs $c$, and goal.

Now we are ready to define the TSP formally.

Definition 2.12. The traveling salesman problem (TSP) is the following optimization problem.

**Input:** A complete weighted graph $G = (V, E, c)$.

**Constraints:** $M(G, c) = \{(v_1, \ldots, v_n, v_1) \mid (v_1, \ldots, v_n) \text{ is a permutation of } V\}$ is the set of all Hamiltonian cycles in $G$.

**Costs:** The cost of a Hamiltonian cycle $H = (v_1, \ldots, v_n, v_1)$ is $c(H) = \sum_{i=1}^{n-1} c(v_i, v_{i+1}) + c(v_n, v_1)$.

**Goal:** Minimum.

The $\beta$-metric traveling salesman problem ($\Delta_\beta$-TSP) is the TSP restricted to $\beta$-metric graphs.

Analogous to the classes $\mathbb{P}$ and $\mathbb{NP}$, there exist classes $\mathbb{PO}$ and $\mathbb{NPO}$ for optimization problems.
Definition 2.13. The class NPO is the set of all optimization problems $U = (\Sigma_I, \Sigma_O, L, M, c, \text{goal})$ satisfying the following conditions:

- $L \in \text{P},$
- there exists a polynomial $p$ such that
  - $|y| \leq p(|x|)$ holds for every $x \in L$ and every $y \in M(x)$ and
  - there exists a polynomial-time algorithm that decides, for every $x \in L$ and every $y \in \Sigma^*_O$ with $|y| \leq p(|x|)$, whether $y \in M(x)$, and
- cost is polynomial-time computable.

Informally, an optimization problem is in NPO if

- one can efficiently determine whether a string is a valid input, i.e., an instance of $U$,
- the solution sizes are polynomial in the input size and one can efficiently determine whether a word is a valid solution to an input, and
- the cost of any solution can be efficiently computed.

Definition 2.14. The class PO is the set of all optimization problems $U = (\Sigma_I, \Sigma_O, L, M, c, \text{goal})$ in NPO for which there exists a polynomial-time algorithm that computes an optimal solution, for every $x \in L$.

Exact algorithms for problems outside PO have exponential time complexity and are therefore rarely useful in practice; hence the focus on approximation algorithms, i.e., algorithms that find an approximate solution in polynomial time. To measure the quality of an approximate solution, we need the following concept.

Definition 2.15. Let $U = (\Sigma_I, \Sigma_O, L, M, c, \text{goal})$ be an optimization problem and let $A$ be an approximation algorithm for $U$. For every $x \in L$, the approximation ratio $R_A(x)$ of $A$ on $x$ is defined as

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

where $Opt_U(x)$ denotes the cost of an optimal solution for $U$ on input $x$.

For $n \in \mathbb{N}$, the approximation ratio of $A$ is defined as

$$R_A(n) = \max \{R_A(x) \mid |x| = n\}.$$

For every $\delta > 1$, $A$ is called $\delta$-approximation algorithm for $U$ if $R_A(x) \leq \delta$, for all $x \in L$. 
Definition 2.16. If the approximation ratio of an algorithm $A$ on all input instances is at most $x$, then we have established an upper bound of $x$ on the approximation ratio of $A$.

On the other hand, if there exists an instance on which $A$ cannot achieve an approximation ratio of $y$, then we have established a lower bound of $y$ on the approximation ratio of $A$.

If the upper and the lower bound coincide (up to an arbitrarily small $\varepsilon > 0$), we say that they are tight.
Chapter 3

The Path Matching Christofides Algorithm

In this chapter, we present the path matching Christofides algorithm, an approximation algorithm for the traveling salesman problem in near-metric graphs that uses the same underlying idea as the well-known Christofides algorithm (Procedure 3.1) and guarantees an upper bound of $\frac{3}{2}\beta^2$ on the approximation ratio. The Christofides algorithm, on the other hand, may return a Hamiltonian cycle that is polynomially worse than the optimal solution when applied to a near-metric graph.

Procedure 3.1 Christofides Algorithm [Chr76]

**Input:** A complete, metric graph $G$.
1: Find a minimum spanning tree $T$ in $G$. Let $U$ be the set of odd vertices in $T$.
2: Find a minimum perfect matching $\Pi$ for $U$ in $G$.
3: Construct an Eulerian cycle $E$ in $T$ and $\Pi$.
4: Transform $E$ into a Hamiltonian cycle $H$ by skipping repeated occurrences of vertices.

**Output:** $H$.

For example, consider the complete graph with vertices $v_0, v_1, \ldots, v_n$ with $n = 2^k$, for some $k \in \mathbb{N}$, with edge lengths as follows. The edges $\{v_i, v_{i+1}\}$, for $i = 0, 1, \ldots, n-1$, have length 1, and all other edges have maximum possible length such that the $\beta$-triangle inequality is not violated. In particular, $c(v_i, v_{i+2^m}) = (2\beta)^m$, for $i = 0, 1, \ldots, n-2^m$ (see Figure 3.1). In step 1, the Christofides algorithm obtains the unique minimum spanning tree with all edges of length 1. Because this tree contains only two odd vertices $v_0$
Chapter 3. The Path Matching Christofides Algorithm

and \( v_n \), these are connected by the direct edge between them in step 2, resulting in the Eulerian cycle \( E = (v_0, v_1, \ldots, v_n, v_0) \) in step 3. Since \( E \) is already a Hamiltonian cycle, step 4 is skipped and \( E \) is the final solution. The length of \( E \) is \( n(\beta^k + 1) \). On the other hand, the optimal solution \( H_{\text{Opt}} = (v_0, v_2, v_4, \ldots, v_n, v_n-1, v_n-3, \ldots, v_1, v_0) \) has length \( 2\beta(n - 1) + 2 \), i.e., on this graph, the Christofides algorithm achieves an approximation ratio of

\[
\frac{\text{cost}(E)}{\text{cost}(H_{\text{Opt}})} = \frac{n(\beta^k + 1)}{2\beta(n - 1) + 2} \geq \frac{\beta^k n}{2\beta n} = \frac{\beta^{k-1}}{2},
\]

which grows polynomially in \( n \).

![Figure 3.1: A worst-case example for the Christofides algorithm.](image)

This example is useful because it demonstrates the following intuitive idea. Let \( p \) be a path in a near-metric graph in which all edges have roughly the same length. Then, an edge connecting the first and the last vertex in \( p \) may have a length polynomial in the number of edges in \( p \). So we have to be careful not to skip too many consecutive vertices.

Böckenhauer et al. [BHK+02] constructed the path matching Christofides algorithm (PMCA, Procedure 3.2) to do just that. The basic idea is the same as for the Christofides algorithm. It also constructs a minimum spanning tree \( T \) and a matching \( \Pi \) for all odd vertices in \( T \), then finds an Eulerian cycle in \( T \) and \( \Pi \), and transforms it into a Hamiltonian cycle. The difference lies in the construction of the matching. As the name implies, the PMCA finds a path matching, i.e., a matching where the connection between two vertices \( u \) and \( v \) is not necessarily the direct edge between them, but some path from \( u \) to \( v \).
3.1 The Algorithm

The PMCA skips at most three consecutive vertices, and thus its approximation ratio has an upper bound of $\frac{3}{2} \beta^2$ [BHK+02].

This chapter is organized as follows. In Section 3.1, we present the algorithm and correct an implementation error from [BHK+02]. To get the reader familiar with the algorithm, we present a family of graphs in Section 3.2 on which it cannot achieve an approximation ratio of $\beta^2 - \varepsilon$, for any $\varepsilon > 0$. In the next chapter, we do the same for $\frac{3}{2} \beta^2 - \varepsilon$, i.e., we show that the upper bound on the approximation ratio is tight.

3.1 The Algorithm

We now present the path matching Christofides algorithm. Let us first introduce its main tools.

A path matching for a vertex set $V$ of even size is a set of $|V|/2$ edge-disjoint paths having the vertices in $V$ as its disjoint endpoints. For simplicity, we address a set of paths and in particular a path matching forming a tree (forest) as a tree (forest). Let $(v_1, v_2, \ldots, v_k)$ be a path. A bypass in it is an edge $\{u, v\}$ replacing a sub-path $(u = v_i, v_{i+1}, \ldots, v_j = v)$, for $1 \leq i < i+1 < j \leq k$. Also, we say that the vertices $v_{i+1}, v_{i+2}, \ldots, v_{j-1}$ are bypassed. The size of a bypass is the number of replaced edges $j - i$.\footnote{We are not interested in bypasses of size 1.} A conflict in a set of paths is a vertex that occurs in more than one path. A conflict in an Eulerian cycle is a vertex that is visited more than once. An inner vertex of a path $(v_1, v_2, \ldots, v_k)$ is any of the vertices $v_2, v_3, \ldots, v_{k-1}$. An inner conflict of a path $p$ is a vertex internal to $p$ that is a conflict.

**Procedure 3.2 Path Matching Christofides Algorithm [BHK+02]**

**Input:** A complete $\beta$-metric graph $G$, for some $\beta \geq 1$.

1. Find a minimum spanning tree $T$ in $G$. Let $U$ be the set of odd vertices in $T$.

2. Construct a minimum path matching $\Pi$ for $U$. (The matching is edge-disjoint as a direct consequence of its minimality.)

3. Resolve conflicts in $\Pi$ to obtain a vertex-disjoint path matching $\Pi'$.

4. Construct an Eulerian cycle $E := (p_1, q_1, p_2, q_2, \ldots)$ on $T$ and $\Pi'$ such that $p_1, p_2, \ldots$ are paths in $T$ and $q_1, q_2, \ldots$ are paths in $\Pi'$.

5. Transform $p_1, p_2, \ldots$ into $p_1', p_2', \ldots$ such that the forest $T_f$ formed by $p_1', p_2', \ldots$ has maximum degree 3. Let $E' := (p_1', q_1, p_2', q_2, \ldots)$.

6. Resolve all remaining conflicts in $E'$ to obtain a Hamiltonian cycle $H$.

**Output:** $H$. 

\footnote{We are not interested in bypasses of size 1.}
Chapter 3. The Path Matching Christofides Algorithm

The implementation of steps 1 and 4 are well-known [CLRS09, GP05]. Note that in order for the edges of each path in \( \Pi \) to remain consecutive in \( E \), each such path has to be regarded as a single edge while computing the Eulerian cycle. The following sections present the implementations of steps 2, 3, 5, and 6 from [BHK+02].

3.1.1 Construction of a Minimum Path Matching

Step 2 can be implemented by constructing a new, complete graph \( G' \) over \( U \) where the length of an edge \( \{v, w\} \) corresponds to the length of a shortest path between \( v \) and \( w \) in \( G \). Then, we compute a minimum perfect matching \( M \) in \( G' \). After that, we map \( M \) back to \( G \) by connecting two vertices in \( G \) via a shortest path if they are matched in \( M \).

**Lemma 3.1 [BHK+02].** The obtained minimum path matching is cycle-free and edge-disjoint.

3.1.2 Conflict Resolution in the Minimum Path Matching

Now we transform the cycle-free, edge-disjoint path matching \( \Pi \) into a vertex-disjoint path matching \( \Pi' \). We do this for every tree in \( \Pi \) separately, processing path by path. We cannot, however, bypass conflicts arbitrarily, because there are two problematic situations.

First, let \( p \) be the path we are currently processing. We do not know what to do if an endpoint \( v \) of \( p \) is a conflict. We cannot bypass \( v \) in \( p \), so the only obvious possibility is that all other paths containing \( v \) bypass it. But this is problematic because of the following.

Second, there can be an arbitrary number of neighboring conflicts in a path. More formally, for a path \( p = (v_1, v_2, \ldots, v_k) \), it is possible that all the vertices \( v_{i+1}, \ldots, v_{j-1} \), for \( 1 < i + 1 \leq j - 1 < k \), are conflicts. The edge \( \{v_i, v_j\} \) bypassing them in \( p \) can have length \( \Theta(\beta^{\lceil \log (j-i) \rceil}) \) (see Figure 3.2). This is essentially the same problem as in the worst-case example for the Christofides algorithm.

Procedure 3.3 is rather lengthy mainly because of these two problems. Roughly speaking, we search for a path with exactly one conflict (such a path exists because \( \Pi \) is cycle-free) and eliminate it until no such path is left.

More precisely, we first pick a tree and a path in it that has at least one inner conflict. (In a tree that still has conflicts, every path has at least one conflict. Otherwise, it would be a tree on its own.) Then, we search for a path that has only one conflict as follows.
3.1. The Algorithm

As long as the path currently being processed contains more than one conflict, we pick the first or the last conflicting vertex in it and then switch to another, not yet visited path that also contains this vertex. We thus always end up in a path \( p \) with only one conflict \( v \).

Procedure 3.3
Input: An edge-disjoint, cycle-free path matching \( \Pi \) for \( U \) in \( G \).

1: for all trees \( \Pi_T \) in \( \Pi \) do
2: while \( \Pi_T \) has conflicts do
3: pick an arbitrary path \( p \in \Pi_T \)
4: if \( p \) has only one conflict \( v \), and \( v \) is an endpoint of \( p \) then
5: pick as new \( p \) another path for which \( v \) is also a conflict
6: end
7: while \( p \) has more than one conflict do
8: let \( u, v \) be the first and the last conflict in \( p \)
9: let \( p_u, p_v \in \Pi_T \) be paths that contain \( u \) resp. \( v \)
10: pick as new \( p \) one of \( p_u, p_v \) that was formerly not picked
11: end
12: let \( v \) be the only conflict in the finally chosen path \( p \)
13: if \( v \) is internal to \( p \) then
14: bypass \( v \) in \( p \)
15: else
16: (a single edge \( e \) is incident to \( v \) in \( p \))
17: bypass \( e \) together with one edge of the previously picked path as shown in Figure 3.3
18: end
19: end
20: end
Output: A vertex-disjoint path matching \( \Pi' = (q_1, q_2, \ldots) \).

If \( v \) is an inner conflict of \( p \), we simply bypass it in \( p \). If \( v \) is an endpoint of \( p \), let \( p' \) be another path containing \( v \). We transform \( p \) and \( p' \) and then
resolve the conflict as shown in Figure 3.3. This simplifies the proof of the second part of Lemma 3.2, which we omit here. In the latter case, observe the following four things.

- The transformation as shown in Figure 3.3 does not change the set of end vertices of $\Pi$.
- Since we always pick the first or the last conflict in a path, there cannot be conflicts on both sides of $v$ in $p'$. That is, in Figure 3.3 there is no conflict in $p'$ below $v$.
- $p'$ contains at least one other conflict, because otherwise, $\Pi_T$ contains only $p$ and $p'$, and no matter with what path we start, we always end up bypassing $v$ in $p'$.
- From the definition of a path matching, it follows that a vertex cannot be an endpoint of two different paths, which in turn guarantees that $v$ is an inner vertex of $p'$.

![Figure 3.3: Conflict resolution between the finally chosen path $p$ and another path $p'$ that has at least one more conflict.](image)

**Lemma 3.2** [BHK+02]. Procedure 3.3 transforms an edge-disjoint, cycle-free path matching into a vertex-disjoint path matching. Moreover, every path in $\Pi'$ has at most one bypass, and every bypass has size 2.

Let us do a short example to see what Procedure 3.3 actually does in practice.

Consider the complete graph $G_{3,8}(\beta)$ with vertices $a_i, b_i, c_i$, for $1 \leq i \leq 8$, edge length 1 for the edges $\{a_i, b_i\}, \{b_i, c_i\}$, and $\{b_i, b_{i+1}\}$, and maximum possible length for all other edges such that the $\beta$-triangle inequality is not violated. It is obvious that the minimum spanning tree consists of all edges...
3.1. The Algorithm

of length 1. Therefore, the odd vertices are all \(a_i\), all \(c_i\), and \(b_1\) and \(b_8\). A possible minimum path matching can be seen in Figure 3.4.

Let us first pick the path \((a_1, b_1)\). Since it has only one conflict \(b_1\), and \(b_1\) is an endpoint, we switch to the path \((c_1, b_1, b_2)\). Since this path still has more than one conflict, we switch to the path \((a_2, b_2, b_3, a_3)\) and so on, until we arrive at the path \((a_8, b_8)\). Now we have the special case as shown in Figure 3.3. We transform the paths \((c_7, b_7, c_8)\) and \((a_8, b_8)\) into \((c_7, b_7, b_8)\) and \((a_8, b_8, c_8)\) and then bypass \(b_8\) in the latter path (see Figure 3.5).

And now it suddenly becomes very easy. We may pick \((c_7, b_7, b_8)\) and bypass \(b_7\), then pick \((a_6, b_6, b_7, a_7)\) and bypass \(b_6\) and so on. In the end, we obtain the vertex-disjoint path matching shown in Figure 3.6.
3.1.3 Conflict Resolution in the Minimum Spanning Tree

Remember that we are now in step 5, meaning we computed in the previous step an Eulerian cycle on \( T \) and \( \Pi' \) that consists of paths \( p_1, p_2, \ldots \) in \( T \) and \( q_1, q_2, \ldots \) in \( \Pi' \). The goal of the current step is to transform the paths \( p_1, p_2, \ldots \) into paths \( p'_1, p'_2, \ldots \) such that the forest \( T_f \) formed by these paths has maximum degree 3. We do this in the following way. First, we pick an arbitrary vertex \( r \) as a root. Then, we bypass the vertex \( v \) closest to \( r \) in each path \( p_i \) if \( v \) is an inner vertex of \( p_i \). We indeed achieve the desired result.

Procedure 3.4 is, however, a little too ambitious. There are inputs for which we drop certain vertices completely, which might not be what we want. We will deal with this problem in Section 3.1.5.

Procedure 3.4

\begin{itemize}
  \item \textbf{Input:} The paths \( p_1, p_2, \ldots \) forming \( T \) computed in step 4 of the PMCA.
  \item 1: pick an arbitrary vertex \( r \in T \)
  \item 2: for all paths \( p_i \) in \( T \) do
  \item 3: let \( v \) be the vertex in \( p_i \) of minimal distance to \( r \) in \( T \)
  \item 4: if \( v \) is internal to \( p_i \) then
  \item 5: bypass \( v \) in \( p_i \) and call this new path \( p'_i \)
  \item 6: end
  \item 7: call the resulting path \( p'_i \)
  \item 8: end
\end{itemize}

\textbf{Output:} The paths \( p'_1, p'_2, \ldots \) forming a forest \( T_f \).

Lemma 3.3 \([BHK+02]\). Every path in \( T_f \) has at most one bypass, and every bypass has size 2. Moreover, every vertex in \( T_f \) has degree at most 3.

Corollary 3.4. Every vertex has degree at most 4 in \( E' := (p'_1, q_1, p'_2, q_2, \ldots) \).

Proof. A vertex \( v \) in \( E' \) has even degree. Therefore assume \( \deg(v) \geq 6 \). Because of Lemma 3.3, \( v \) must then be lying on at least two paths \( q_i, q_j \) in \( \Pi' \). This contradicts Lemma 3.2.

3.1.4 Conflict Resolution in the Modified Eulerian Cycle

We are now in the last step of the PMCA, where we transform an Eulerian cycle in which every vertex is visited at most twice into a Hamiltonian cycle. How we do this is crucial. Let us reconsider our worst-case example for the
3.1. The Algorithm

Christofides algorithm for, say, $n = 8$. After step 5, we may have computed the Eulerian cycle $(v_0, v_1, \ldots, v_8, v_7, \ldots, v_0)$. If we just arbitrarily skip vertices now, we can still end up in the same bad solution that the Christofides algorithm computes. (See Figure 3.7, where the dashed path is the only path in $\Pi'$ and the solid path the only path in $T_f$.)

![Figure 3.7](image)

Figure 3.7: Arbitrarily skipping vertices in step 6 can yield a bad solution.

**Procedure 3.5**

**Input:** The Eulerian cycle $E'$ computed in step 5 of the PMCA, in which every vertex occurs at most twice (because of Corollary 3.4).

1: take an arbitrary conflict, i.e., a vertex occurring twice as $u$ and $u'$ in $E'$
2: bypass $u$
3: while there are conflicts remaining do
4: if occurrence $u$ has at least one unresolved conflict as neighbor then
5: let $v$ be one of them, chosen by Rule 3.5
6: resolve that conflict by bypassing the other occurrence of $v$
7: else
8: resolve an arbitrary conflict
9: end
10: let $u$ be the newly bypassed vertex occurrence
11: end

**Output:** The Hamiltonian cycle $H$.

**Rule 3.5.** If between $u$ and another bypassed vertex occurrence $t$ in $E'$, there are only unresolved conflicts, choose $v$ to be the neighbor of $u$ towards $t$.

**Lemma 3.6** [BHK$^+02$]. Procedure 3.5 resolves all conflicts by generating bypasses of total size at most 4. $\square$

The basic idea of Procedure 3.5 is very simple. After bypassing a vertex occurrence $u$, we check if either of its neighbor occurrences is a conflict. If this is the case, we pick one of them, say $v$. Now $v$ occurs once more in $E'$, and we
just bypass this other occurrence. With this simple trick, the PMCA avoids computing polynomially long edges and, e.g., returns the optimal solution in the example above.

There is just one problem. We might obtain edges of length $\Theta(\beta^3)$ if we always pick the wrong neighbor; hence Rule 3.5. (Its unambiguity is proved in [BHK+02].) We will not need this rule in our examples of hard input instances, but nevertheless present an example to show its necessity in general.

Consider the complete graph $G(\beta)$ with vertices $a_i, b_i$, for $1 \leq i \leq 6$, and $c_j$, for $1 \leq j \leq 4$, and edge lengths as follows. The edges shown in Figure 3.8 have length 1, and all other edges have maximum possible length such that the $\beta$-triangle inequality is not violated.

The edges of length 1 form a minimum spanning tree in $T$, resulting in the set of odd vertices $U = \{a_1, a_3, a_4, a_5, a_6, b_1, b_2, b_6, c_1, c_2, c_3, c_4\}$.

The PMCA might now compute the minimum path matching

$$\Pi = \{(b_1, b_2), (a_1, a_2, b_2, b_3, b_4, b_5, b_6)\} \cup \{(a_i, b_{i-2}) \mid 3 \leq i \leq 6\}$$

for $U$ and resolve the conflicts in it to obtain the vertex-disjoint path matching

$$\Pi' = \{(b_1, b_2), (a_1, a_2, b_3, b_4, b_5, b_6)\} \cup \{(a_i, c_{i-2}) \mid 3 \leq i \leq 6\}$$

as depicted in Figure 3.9.

One possible Eulerian cycle on $T$ and $\Pi'$ is

$$E = (b_1, b_2, b_3, a_3, a_1, b_1, b_2, a_4, a_5, a_6, b_6, b_5, b_4, b_3, a_2, a_1, a_2, b_2, b_1)$$

as shown in Figure 3.10. For clarity, the paths of $\Pi'$ are dashed.

The Eulerian cycle $E$ can be seen as a sequence of paths $p_1, q_1, p_2, q_2, \ldots$, where $p_i$ are paths consisting of edges from $T$ and $q_i$ are paths in $\Pi'$. Let $p_1 := (b_2, b_3, a_3), p_2 := (c_1, b_3, b_4, c_2), p_3 := (a_4, b_4, b_5, a_5), p_4 := (c_3, b_5, b_6, c_4), p_5 := (a_6, b_6), p_6 := (a_1, a_2, b_2, b_1).
3.1. The Algorithm

The goal of the next step is to obtain an Eulerian cycle in which at most three edges from $T$ are adjacent to every vertex. This is already the case for all vertices except $b_3, b_4,$ and $b_5$. The PMCA chooses $r := b_1$ and obtains the modified Eulerian cycle

$$E' = (b_1, b_2, b_3, a_3, c_1, b_4, c_2, a_4, a_5, a_6, b_5, b_4, b_3, a_2, a_1, a_2, b_2, b_1)$$

as shown in Figure 3.11 by bypassing $b_3, b_4,$ and $b_5$ in $p_2, p_3,$ and $p_4$, respectively.

Now we start with the conflict resolution in $E'$, i.e., with step 6. We bypass $b_3$ between $b_4$ and $a_2, a_2$ between $a_1$ and $b_2, b_2$ between $b_1$ and $b_3,$ and $b_5$ between $b_6$ and $b_1$. The resulting graph can be seen in Figure 3.12.

Remember that we do this example to show that, without the use of Rule 3.5, we could run into trouble in the last step, i.e., we might obtain
edges of length $\Theta(\beta^3)$ after the conflict resolution in $E'$. We first show what happens if we ignore Rule 3.5 and afterwards what happens if we apply it.

If we ignore Rule 3.5, we can arbitrarily choose what conflict to resolve next. Let us choose $b_6$ and bypass $b_6$ between $c_3$ and $c_4$. The only remaining conflict is $b_4$, which we bypass between $b_6$ and $a_2$. The resulting Hamiltonian cycle as shown in Figure 3.13 contains the edge $\{a_2, b_6\}$ of length $\Theta(\beta^3)$.

On the other hand, if we apply Rule 3.5, we must resolve $b_4$ after bypassing $b_5$ because between the bypassed occurrence of $b_5$ and the previously bypassed occurrence of $b_3$ there are only conflicting vertices (namely $b_4$). That is, we bypass $b_4$ between $c_1$ and $c_2$, and then finally bypass $b_6$ e.g. between $c_3$ and $c_4$. The resulting Hamiltonian cycle as shown in Figure 3.14 contains no edge of length $\Theta(\beta^3)$.
3.1.5 Completeness of the Solution

Now we will show that the solution is complete. By this we mean that no vertex is dropped at some point during the execution, i.e., the returned Hamiltonian cycle \( H \) contains every vertex in \( V \) at least once.\(^2\) A vertex can only be dropped in steps 5 and 6. So we need to show that these steps do not drop a vertex in such a way that it is no longer contained in the solution. Unfortunately, with the current implementation of step 5, a vertex might actually be dropped completely. Consider the following scenario.

Let \( G := K_3 \) with vertices \( a, b, c \) and edge lengths \( c(a, b) = c(b, c) := 1, c(a, c) := 2 \). After step 4 of the PMCA, we may have \( p_1 = (a, b, c), q_1 = (c, a) \). If Procedure 3.4 now chooses \( r := b \), it transforms \( p_1 \) into \( p'_1 = (a, c) \), and thus we have \( E' = (p'_1, q_1) = (a, c, a) \) after step 5 of the PMCA. Because there are no more conflicts, step 6 also returns \( (a, c, a) \). So the solution does not contain the vertex \( b \) and is therefore clearly no Hamiltonian cycle. The problem is that Procedure 3.4 picks a root \( r \) and then bypasses the vertex in every path \( p_i \) that is closest to \( r \) independent of whether that is actually necessary.

Procedure 3.6 takes into account that the goal of step 5 is merely to obtain a forest \( T_f \) that contains only vertices of degree at most 3 and thus strengthens the condition to bypass a vertex.

\[\text{Procedure 3.6}\]
\[
\text{Input:} \quad \text{The paths } p_1, p_2, \ldots \text{ forming } T \text{ computed in step 4 of the PMCA.}
\]
\[\begin{align*}
1: & \quad \text{pick an arbitrary vertex } r \in T \\
2: & \quad T_f := T \\
3: & \quad \textbf{for all paths } p_i \text{ in } T \textbf{ do}
4: & \quad \text{let } v \text{ be the vertex in } p_i \text{ of minimal distance to } r \text{ in } T \\
5: & \quad \textbf{if } v \text{ is internal to } p_i \text{ and } \deg_{T_f}(v) \geq 4 \textbf{ then}
6: & \quad \text{bypass } v \text{ in } p_i \text{ and call this new path } p'_i \\
7: & \quad \textbf{end}
8: & \quad \text{call the resulting path } p'_i \\
9: & \quad T_f := T_f - p_i + p'_i \\
10: & \quad \textbf{end}
\]
\[\text{Output:} \quad \text{The paths } p'_1, p'_2, \ldots \text{ forming a forest } T_f.\]

Let us see what happens when we apply the PMCA on \( K_3 \) as defined above with Procedure 3.6 instead of Procedure 3.4. Since the first four steps

\[H \text{ contains every vertex in } V \text{ at most once because that is the termination condition of the while-loop in Procedure 3.5.}\]
are the same, we may still have \( p_1 = (a, b, c), q_1 = (c, a) \) after step 4. Let us again assume that the vertex \( b \) is chosen as \( r \). Thanks to the strengthened condition in line 5, line 6 is skipped and thus \( p'_1 := p_1 \). So step 5 of the PMCA returns \( p'_1 = (a, b, c) \), and because there are no more conflicts, step 6 returns \( (p'_1, q_1) = (a, b, c, a) \), which is indeed a Hamiltonian cycle in \( K_3 \).

We now show the completeness of steps 5 and 6.

**Lemma 3.7.** Procedure 3.6 returns a forest that contains all vertices in \( V \).

*Proof.* Assume for contradiction that there is a vertex \( v \) that is not in the forest. Then, \( v \) must have been dropped in line 6 at some point in time. But this is only possible if \( v \) had then still degree at least 4, i.e., there exists another path that still contains \( v \). \( \square \)

**Corollary 3.8.** \( E' \) contains all vertices in \( V \).

**Lemma 3.9.** Procedure 3.5 returns a Hamiltonian cycle \( H \) that contains all vertices in \( E' \).

*Proof.* Assume for contradiction that there is a vertex \( v \) that is not in \( H \). Then, \( v \) must have been dropped in line 6 or 8 at some point in time. But this is only possible if \( v \) was then still visited exactly once more in \( E' \), i.e., it is still contained in \( H \). \( \square \)

From the observations above immediately follows that \( H \) contains every vertex in \( V \) at least once.

### 3.2 An Example

To understand the PMCA better, we now apply it to a family of graphs and show that, for every \( \beta \geq 1 \), it cannot achieve an approximation ratio of \( \beta^2 - \varepsilon \), for any \( \varepsilon > 0 \), on this family.

Let \( G_{4,k}(\beta) \) be the complete graph with the vertices \( \{a_i, b_i, c_i, d_i \mid 1 \leq i \leq k\} \), for even \( k \in \mathbb{N} \), with edge lengths

\[
\begin{align*}
c(a_i, b_i) &= c(b_i, b_j) = c(c_i, d_i) := 1/k, \quad c(a_i, a_j) := 3/k, \\
c(b_i, c_i) &= c(c_i, c_j) := 1, \quad c(d_i, d_j) := 1 + 2/k,
\end{align*}
\]

for \( i \neq j \), and maximum possible length for all other edges such that the \( \beta \)-triangle inequality is not violated.

Let \( A = \{a_i \mid 1 \leq i \leq k\} \), \( B = \{b_i \mid 1 \leq i \leq k\} \), \( C = \{c_i \mid 1 \leq i \leq k\} \), and \( D = \{d_i \mid 1 \leq i \leq k\} \).

Figure 3.15 shows the basic grid structure of the graph. (Only some edges are shown.)
3.2. An Example

Lemma 3.10. The graph $G_{4,k}(\beta)$ satisfies the $\beta$-triangle inequality.

Proof. Let $\{u, v, w\}$ be a triangle, and let $e = \{u, v\}$ be the longest edge in it, i.e., the one that could be responsible for a violation. (At most one edge per triangle can violate the $\beta$-triangle inequality.) We show that there always exists another edge in this triangle that is long enough such that $e$ does not violate the $\beta$-triangle inequality.

The edge $e$ can, by construction, only be one of those edges for which we explicitly defined the edge length. Furthermore, all edges for which we did not explicitly define the length have length at least $2/k$. All edges in the entire graph thus have length at least $1/k$, and we have to prove the statement only in the following four cases.

Case 1: $e = \{a_i, a_j\}$.

If $w \in A$, then all three edges in the triangle $\{u, v, w\}$ have length $3/k$. Thus $c(a_i, a_j) = 3/k \leq \beta \cdot 6/k = \beta(c(a_i, w) + c(w, a_j))$.

If $w = b_i$, then a shortest path from $a_j$ to $w$ is $(a_j, b_i, b_i)$, which has length $2/k$. Thus, the edge $\{a_j, w\}$ also has length at least $2/k$. If $w = b_i$ for some $l \neq i$, then a shortest path from $a_i$ to $w$ is $(a_i, b_i, w)$, which has length $2/k$. Thus, the edge $\{a_i, w\}$ also has length at least $2/k$. In both cases, $c(a_i, a_j) = 3/k \leq \beta \cdot 3/k \leq \beta(c(a_i, w) + c(w, a_j))$.

If $w = c_i$, then a shortest path from $a_j$ to $w$ is $(a_j, b_i, b_i, c_i)$, which has length $1 + 2/k$. Thus, the edge $\{a_j, w\}$ also has length at least $1 + 2/k$. If $w = c_i$ for some $l \neq i$, then a shortest path from $a_i$ to $w$ is $(a_i, b_i, b_i, c_i)$, which has length $1 + 2/k$. Thus, the edge $\{a_i, w\}$ also has length at least $1 + 2/k$. In both cases, $c(a_i, a_j) = 3/k \leq \beta(1 + 3/k) \leq \beta(c(a_i, w) + c(w, a_j))$.

If $w = d_i$, then a shortest path from $a_j$ to $w$ is $(a_j, b_i, b_i, c_i, w)$, which has length $1 + 3/k$. Thus, the edge $\{a_j, w\}$ also has length at least $1 + 3/k$. If $w = d_i$ for some $l \neq i$, then a shortest path from $a_i$ to $w$ is $(a_i, b_i, b_i, c_i, w)$, which has length $1 + 3/k$. Thus, the edge $\{a_i, w\}$ also has length at least $1 + 3/k$. In both cases, $c(a_i, a_j) = 3/k \leq \beta(1 + 4/k) \leq \beta(c(a_i, w) + c(w, a_j))$.
Case 2: \( e = \{b_i, c_i\} \). If \( w \in A \cup B \), then the shortest path from \( c_i \) to \( w \) has length at least \( 1 + 1/k \), and thus also the edge \( \{c_i, w\} \). Therefore, \( c(b_i, c_i) = 1 \leq \beta(1 + 2/k) \leq \beta(c(c_i, w) + c(w, b_i)) \). Otherwise, the shortest path from \( w \) to \( b_i \) has length at least \( 1 + 1/k \), and thus also the edge \( \{w, b_i\} \). Therefore, \( c(b_i, c_i) = 1 \leq \beta(1 + 2/k) \leq \beta(c(c_i, w) + c(w, b_i)) \).

Case 3: \( e = \{c_i, c_j\} \). If \( w = b_i \) or \( w = d_i \), then the shortest path from \( c_j \) to \( w \) has length at least \( 1 + 1/k \), and thus also the edge \( \{c_j, w\} \). Therefore, \( c(c_i, c_j) = 1 \leq \beta(1 + 2/k) \leq \beta(c(c_i, w) + c(w, c_j)) \). Otherwise, the shortest path from \( w \) to \( c_i \) has length at least 1, and thus also the edge \( \{w, c_i\} \). Therefore, \( c(c_i, c_j) = 1 \leq \beta(1 + 1/k) \leq \beta(c(c_j, w) + c(w, c_i)) \).

Case 4: \( e = \{d_i, d_j\} \). If \( w = c_i \), then the shortest path from \( d_j \) to \( w \) has length at least \( 1 + 1/k \), and thus also the edge \( \{d_j, w\} \). Therefore, \( c(d_i, d_j) = 1 + 2/k \leq \beta(1 + 2/k) \leq \beta(c(d_j, w) + c(w, d_i)) \). Otherwise, the shortest path from \( w \) to \( d_i \) has length at least \( 1 + 1/k \), and thus also the edge \( \{w, d_i\} \). Therefore, \( c(d_i, d_j) = 1 + 2/k \leq \beta(1 + 2/k) \leq \beta(c(d_j, w) + c(w, d_i)) \).

Figure 3.16 shows a Hamiltonian cycle in \( G_{4,k}(\beta) \).

It contains \( k \) edges from a vertex in \( A \) to a vertex in \( B \) of length \( 3/k \) each and another \( k \) edges from a vertex in \( C \) to a vertex in \( D \) of length \( 1/k \) each. There are the two edges \( \{b_1, c_1\}, \{b_k, c_k\} \) of length 1 each. The length of all “vertical” edges thus amounts to 4.

There are \( k/2 \) edges between vertices in \( A \) as well as \( k/2 \) edges between vertices in \( D \), the former of length \( 3/k \) each and the latter of length \( 1 + 2/k \) each. There are \( k/2 - 1 \) edges between vertices in \( B \) and \( k/2 - 1 \) edges between vertices in \( C \), the former of length \( 1/k \) each and the latter of length 1 each. The length of all “horizontal” edges thus amounts to \( k + 2 - 1/k \).

The depicted Hamiltonian cycle thus has length \( k + 6 - 1/k \).

![Figure 3.16: A Hamiltonian cycle of length k + 6 − 1/k in G_{4,k}(\beta).](image-url)

We now show one possible implementation of the PMCA that, on input \( G_{4,k}(\beta) \), returns a Hamiltonian cycle of length \( (k - 2)\beta^2 \). In many steps, the
3.2. An Example

algorithm can choose between different possibilities. We do not discuss this in every step, but just describe what our implementation chooses to do.

3.2.1 Minimum Spanning Tree

The PMCA computes the minimum spanning tree \( T \) with edges \( \{\{a_i, b_i\}, \{b_i, c_i\}, \{c_i, d_i\} \mid 1 \leq i \leq k\} \cup \{\{b_i, b_{i+1}\} \mid 1 \leq i \leq k-1\} \)
resulting in the set of odd vertices
\[
U = \{a_i, d_i \mid 1 \leq i \leq k\} \cup \{b_1, b_k\}
\]
as depicted in Figure 3.17.

\[
\begin{array}{ccccccc}
  a_1 & a_2 & a_3 & \cdots & a_k \\
  b_1 & b_2 & b_3 & \cdots & b_k \\
  c_1 & c_2 & c_3 & \cdots & c_k \\
  d_1 & d_2 & d_3 & \cdots & d_k \\
\end{array}
\]

Figure 3.17: A minimum spanning tree in \( G_{4,k}(\beta) \). The odd vertices are circled.

It is easy to see that \( T \) is indeed a minimum spanning tree. Every edge in the graph has length at least 1/k, therefore the edges \( \{a_i, b_i\}, \) for \( 1 \leq i \leq k, \)
and \( \{b_j, b_{j+1}\}, \) for \( 1 \leq j \leq k - 1, \) form a minimum spanning tree for the vertex set \( A \cup B. \) On the other hand, the edges \( \{c_i, d_i\}, \) for \( 1 \leq i \leq k, \) form minimum spanning trees for the respective vertex sets. All we need to do is add an edge for every component \( \{c_i, d_i\} \) such that it is connected to \( A \cup B \) in the end. All outgoing edges of a component \( \{c_i, d_i\} \) have length at least 1, so we can just take the edges \( \{b_i, c_i\}, \) for \( 1 \leq i \leq k. \)

3.2.2 Minimum Path Matching

The PMCA computes the minimum path matching
\[
\Pi = \{(a_1, b_1)\} \cup \{(a_{2i}, b_{2i}, b_{2i+1}, a_{2i+1}) \mid 1 \leq i < k/2 - 1\} \\
\cup \{(a_k, b_k)\} \cup \{(d_{2i-1}, c_{2i-1}, c_{2i}, d_{2i}) \mid 1 \leq i \leq k/2\}
\]
as shown in Figure 3.18.
Chapter 3. The Path Matching Christofides Algorithm

\begin{figure}
\centering
\begin{tikzpicture}
  \node (a) at (0,0) {$a_1$};
  \node (b) at (1,0) {$a_2$};
  \node (c) at (2,0) {$a_3$};
  \node (d) at (3,0) {$a_k$};
  \node (e) at (0,-1) {$b_1$};
  \node (f) at (1,-1) {$b_2$};
  \node (g) at (2,-1) {$b_3$};
  \node (h) at (3,-1) {$b_k$};
  \node (i) at (0,-2) {$c_1$};
  \node (j) at (1,-2) {$c_2$};
  \node (k) at (2,-2) {$c_3$};
  \node (l) at (3,-2) {$c_k$};
  \node (m) at (0,-3) {$d_1$};
  \node (n) at (1,-3) {$d_2$};
  \node (o) at (2,-3) {$d_3$};
  \node (p) at (3,-3) {$d_k$};
  \draw (a) -- (b) -- (c) -- (d);
  \draw (e) -- (f) -- (g) -- (h);
  \draw (i) -- (j) -- (k) -- (l);
  \draw (m) -- (n) -- (o) -- (p);
\end{tikzpicture}
\caption{A minimum path matching for $U$ in $G_{4,k}(\beta)$.}
\end{figure}

**Theorem 3.11.** $\Pi$ is a minimum path matching for $U$ in $G_{4,k}(\beta)$, for $k \geq 6$.

In order to prove this result, we need three lemmas. We call the vertices from $U$ in $A$ and $B$ upper vertices and those in $D$ lower vertices.

**Lemma 3.12.** Let $k \geq 6$. In a minimum path matching $\Pi$ for $U$, no path connects an upper and a lower vertex.

*Proof.* Assume for contradiction that there is a path $\pi = (u, \ldots, l)$ in $\Pi$ for some upper vertex $u$ and some lower vertex $l$. Because both the number of upper and lower vertices are even, there must be another such path $\pi' = (u', \ldots, l')$ in $\Pi$, again for some upper vertex $u'$ and some lower vertex $l'$. Observe that $c(\pi) + c(\pi') \geq 2$. We can replace $\pi$ and $\pi'$ with shortest paths $(u, \ldots, u')$ and $(l, \ldots, l')$ of length at most $1 + 5/k$. We know that $k \geq 6$ and thus $1 + 5/k < 2$, i.e., $\Pi$ is no minimum path matching. \qed

Now we can simplify things considerably by splitting up the theorem.

**Lemma 3.13.** The path matching

\[
\{(a_1, b_1), (a_k, b_k)\} \cup \{(a_{2i}, b_{2i}, b_{2i+1}, a_{2i+1}) \mid 1 \leq i < k/2\}
\]

is a minimum path matching for the upper vertices.

*Proof.* The minimum distance between $b_1$ and $b_k$ is $1/k$, which is just the length of the direct path $(b_1, b_k)$ between them. The minimum distance between any two $a_i, a_j$ is $3/k$, so we can either take the direct path $(a_i, a_j)$ or the path $(a_i, b_i, b_j, a_j)$ of the same length. In the implementation we consider, the algorithm always picks paths of the latter form.

There are only two possible forms a matching can have.
3.2. An Example

The first one is that $b_1$ and $b_k$ are matched with vertices from $A$. That is, the matching has the form
\[
\{(a_{i_1}, \ldots, a_{i_2}), (a_{i_3}, \ldots, a_{i_4}), \ldots, (a_{i_{k-3}}, \ldots, a_{i_{k-2}}), (a_{i_{k-1}}, \ldots, b_1), (a_{i_k}, \ldots, b_k)\},
\]
where $(i_1, i_2, \ldots, i_k)$ is a permutation of $(1, 2, \ldots, k)$. In other words, we have a shortest path from some vertex in $A$ to $b_1$, a shortest path from some other vertex in $A$ to $b_k$, and $k/2 - 1$ shortest paths between the remaining vertices in $A$.

In this case, the minimum length is achieved if we set $i_{k-1} := 1, i_k := k$, i.e., if we match $a_1$ with $b_1$ and $a_k$ with $b_k$. The paths $(a_1, b_1)$ and $(a_k, b_k)$ have length $1/k$ each. All other paths have length $3/k$, i.e., the matching has overall length $2 \cdot 1/k + (k/2 - 1) \cdot 3/k = 3/2 - 1/k$.

The other possibility is that $b_1$ and $b_k$ are matched with each other. That is, the matching has the form
\[
\{(a_{i_1}, \ldots, a_{i_2}), (a_{i_3}, \ldots, a_{i_4}), \ldots, (a_{i_{k-3}}, \ldots, a_{i_{k-2}}), (a_{i_{k-1}}, \ldots, a_{i_k}), (b_1, b_k)\},
\]
where again $(i_1, i_2, \ldots, i_k)$ is a permutation of $(1, 2, \ldots, k)$. In other words, we have a shortest path from $b_1$ to $b_k$ and $k$ shortest paths between vertices in $A$.

The path $(b_1, b_k)$ has length $1/k$. All other paths have length $3/k$, i.e., the matching has overall length $1/k + k/2 \cdot 3/k = 3/2 + 1/k$. \hfill \Box

Lemma 3.14. The path matching
\[
\{(d_{2i-1}, c_{2i-1}, c_{2i}, d_{2i}) \mid 1 \leq i \leq k/2\}
\]
is a minimum path matching for the lower vertices.

Proof. The minimum distance between any two $d_i, d_j$ is $1 + 2/k$, which is just the length of the path $(d_i, c_i, c_{i+1}, d_{i+1})$. \hfill \Box

3.2.3 Conflict Resolution in the Minimum Path Matching

The PMCA skips this step because there are no conflicts in $\Pi$.

3.2.4 Eulerian Cycle

The PMCA computes the Eulerian cycle
\[
E = (a_1, b_1, c_1, d_1, c_1, d_1, c_2, d_2, c_2, d_2, c_2, b_2, b_2, b_3, a_3, b_3, \ldots, c_k, d_k, c_k, b_k, a_k, b_k, b_{k-1}, \ldots, b_1, a_1)
\]
as depicted in Figure 3.19. For clarity, the paths of $\Pi$ are dashed.
Chapter 3. The Path Matching Christofides Algorithm

3.2.5 Conflict Resolution in the Minimum Spanning Tree

The goal of the conflict resolution in the minimum spanning tree is that every vertex is adjacent to at most three edges from $T$. Let $r := b_1$.

The vertices $b_i$, for $2 \leq i \leq k - 1$, are the only ones that are adjacent to four edges from $T$, so the PMCA bypasses them between the respective $a_i$ and $c_i$ to obtain the modified Eulerian cycle

$$E' = (a_1, b_1, c_1, d_1, c_1, e_2, d_2, c_2, a_2, b_2, c_3, a_3, c_3, \ldots, c_k, d_k, c_k, b_k, a_k, b_{k-1}, \ldots, b_1, a_1)$$

as shown in Figure 3.20.

---

3. Remember that the choice of $r$ is arbitrary.
3.2.6 Conflict Resolution in the Modified Eulerian Cycle

The goal of the last step is to bypass vertices such that every vertex has degree 2, i.e., we have a Hamiltonian cycle. The problematic vertices are thus all \( b_i \) and \( c_i \).

The PMCA obtains the final Hamiltonian cycle

\[
H = (a_1, b_1, d_1, c_1, c_2, d_2, a_2, b_2, a_3, d_3, \ldots, \\
d_{k-1}, c_{k-1}, d_k, c_k, a_k, b_k, b_{k-2}, \ldots, b_2, a_1)
\]

as depicted in Figure 3.21 by doing the following:

- it bypasses \( c_1 \) between \( b_1 \) and \( d_1 \), \( c_k \) between \( c_{k-1} \) and \( d_k \), and every other \( c_i \) between \( a_i \) and \( d_i \);
- it bypasses \( b_1 \) between \( b_2 \) and \( a_1 \) and \( b_k \) between \( c_k \) and \( a_k \);
- it bypasses \( b_{2i} \) between \( a_{2i} \) and \( b_{2i+1} \), for \( 1 \leq i < k/2 \);
- it bypasses \( b_{2i+1} \) between \( b_{2i+2} \) and \( b_{2i} \), for \( 1 \leq i < k/2 \).

\[\text{cost}(H) \geq (k - 2)\beta^2.\]

Considering only the edges \( \{a_i, d_i\} \), for \( 2 \leq i \leq k - 1 \), we obtain \( \text{cost}(H) \geq (k - 2)\beta^2 \). We have thus shown that, for every \( \beta \geq 1 \) and arbitrarily small \( \varepsilon > 0 \), there exists an implementation \( I \) of the PMCA such that

\[
\frac{\text{cost}(I(G_{4,k}(\beta)))}{\text{Opt}_{\Delta,\text{TSP}}(G_{4,k}(\beta))} \geq \frac{(k - 2)\beta^2}{k + 6 - 1/k} \geq \beta^2 - \varepsilon,
\]

for sufficiently large \( k \), i.e., we have established a lower bound of \( \beta^2 - \varepsilon \) on the approximation ratio of the PMCA.
Chapter 3. The Path Matching Christofides Algorithm
Chapter 4

On the Approximation Ratio of the PMCA

In this chapter, we show that the upper bound of \( \frac{3}{2} \beta^2 \) on the approximation ratio of the PMCA established by Böckenhauer et al. [BHK+02] is tight, i.e., that the following holds.

**Theorem 4.1.** For every \( \beta \geq 1 \), there exists an infinite family of graphs satisfying the \( \beta \)-triangle inequality on which the PMCA cannot achieve an approximation ratio of \( \frac{3}{2} \beta^2 - \varepsilon \), for any \( \varepsilon > 0 \).

The chapter is organized in exactly the same way as Section 3.2. First we introduce the graph and prove that it satisfies the \( \beta \)-triangle inequality. Then, we show that this graph contains a Hamiltonian cycle of a certain length. Finally, we present one possible implementation of the PMCA in order to obtain the desired lower bound.

Let \( G_{10,k}(\beta) \) be the complete graph with the vertices \( \{v_{i,j} \mid 1 \leq i \leq 10, 1 \leq j \leq k\} \), for \( k \in \mathbb{N} \), with edge lengths

- \( c(v_{1,i}, v_{2,i}) = c(v_{2,i}, v_{5,i}) = c(v_{3,i}, v_{5,i}) = c(v_{4,i}, v_{5,i}) = \frac{1}{k} \),
- \( c(v_{5,i}, v_{9,i}) = c(v_{2,i}, v_{9,i}) := 1 \),
- \( c(v_{1,i}, v_{7,i}) := 1 + 2/k \),

for \( i \neq j \), and maximum possible length for all other edges such that the \( \beta \)-triangle inequality is not violated.

Figure 4.1 shows the basic structure of the graph. (Only some edges are shown.) Observe that the graph consists of \( k \) clusters, each consisting of ten vertices. We denote these clusters by \( C_i \), i.e., \( C_i := \{v_{1,i}, v_{2,i}, \ldots, v_{10,i}\} \), for \( 1 \leq i \leq k \). Furthermore, we call the set \( \{v_{1,i}, v_{2,i}, v_{4,i}, v_{5,i}, v_{8,i}, v_{9,i}, v_{10,i}\} \) the upper subcluster \( i \), denoted \( USC_i \), and the vertex set \( \{v_{3,i}, v_{6,i}, v_{7,i}\} \) the lower
Chapter 4. On the Approximation Ratio of the PMCA

subcluster \( i \), denoted \( LSC_i \). The vertices are called USC vertices and LSC vertices, respectively.

The graph \( G_{10,k}(\beta) \) satisfies the \( \beta \)-triangle inequality.

**Proof.** Let \( \{u, v, w\} \) be a triangle, and let \( e = \{u, v\} \) be the longest edge in it, i.e., the one that could be responsible for a violation. We show that there always exists another edge in this triangle that is long enough such that \( e \) does not violate the \( \beta \)-triangle inequality.

The edge \( e \) can, by construction, only be one of those edges for which we explicitly defined the edge length. Furthermore, all edges for which we did not explicitly define the length have length at least \( 2/k \). All edges in the entire graph thus have length at least \( 1/k \), and we have to prove the statement only in the following three cases.

**Case 1:** \( e = \{v_{2,i}, v_{9,j}\} \). If \( w \in C_i \), then the shortest path from \( w \) to \( v_{9,j} \) has length at least \( 1 + 1/k \), and thus also the edge \( \{w, v_{9,j}\} \). Therefore, 
\[
c(v_{2,i}, v_{9,j}) = 1 \leq \beta(1 + 2/k) \leq \beta(c(v_{2,i}, w) + c(w, v_{9,j}))
\]
If \( w \notin C_i \), then the shortest path from \( v_{2,i} \) to \( w \) has length at least \( 1 + 1/k \), and thus also the edge \( \{v_{2,i}, w\} \). Therefore, 
\[
c(v_{2,i}, v_{9,j}) = 1 \leq \beta(1 + 1/k) \leq \beta(c(v_{2,i}, w) + c(w, v_{9,j}))
\]

**Case 2:** \( e = \{v_{5,i}, v_{6,i}\} \). If \( w \in USC_i \), then the shortest path from \( w \) to \( v_{6,i} \) has length at least \( 1 + 1/k \), and thus also the edge \( \{w, v_{6,i}\} \). Therefore, 
\[
c(v_{5,i}, v_{6,i}) = 1 \leq \beta(1 + 2/k) \leq \beta(c(v_{5,i}, w) + c(w, v_{6,i}))
\]
If \( w \notin USC_i \), then the shortest path from \( v_{5,i} \) to \( w \) has length at least \( 1 + 1/k \), and thus also the edge \( \{v_{5,i}, w\} \). Therefore, 
\[
c(v_{5,i}, v_{6,i}) = 1 \leq \beta(1 + 2/k) \leq \beta(c(v_{5,i}, w) + c(w, v_{6,i}))
\]

**Case 3:** \( e = \{v_{1,i}, v_{7,j}\} \). If \( w \in C_i \), then the shortest path from \( w \) to \( v_{7,j} \) has length at least \( 1 + 3/k \), and thus also the edge \( \{w, v_{7,j}\} \). Therefore, 
\[
c(v_{1,i}, v_{7,j}) = 1 + 2/k \leq \beta(1 + 4/k) \leq \beta(c(v_{1,i}, w) + c(w, v_{7,j}))
\]
If \( w \notin C_i \), then the shortest path from \( v_{1,i} \) to \( w \) has length at least \( 1 + 1/k \), and thus also the edge \( \{v_{1,i}, w\} \). Therefore, 
\[
c(v_{1,i}, v_{7,j}) = 1 + 2/k \leq \beta(1 + 2/k) \leq \beta(c(v_{1,i}, w) + c(w, v_{7,j})) \]
\]
\[\square\]
4.1 Minimum Spanning Tree

Figure 4.2 shows a Hamiltonian cycle in $G_{10,k}(\beta)$. In this cycle, every path internal to a cluster, i.e., a path of the form $(v_{1,i}, v_{2,i}, v_{4,i}, v_{8,i}, v_{9,i}, v_{10,i}, v_{5,i}, v_{6,i}, v_{3,i}, v_{7,i})$, for $1 \leq i \leq k$, has length

$$
\frac{1}{k} + \frac{2\beta}{k} + \frac{\beta(1/k + 2\beta/k)}{k} + \frac{1}{k} + \frac{1}{k} + \frac{2\beta/k}{k} + \frac{2\beta}{k} + 1 + \frac{1}{k} + \frac{2\beta}{k} + 1 + \frac{1}{k} + \frac{2\beta}{k} = 1 + \frac{2\beta^2}{k} + \frac{7\beta}{k} + \frac{6}{k}.
$$

These internal paths are connected to each other via $k$ edges of length $1 + 2/k$. So the length of the depicted Hamiltonian cycle is

$$ k \cdot \left(1 + \frac{2\beta^2}{k} + \frac{7\beta}{k} + \frac{6}{k}\right) = 2k + \frac{2\beta^2}{k} + 7\beta + 6. $$

We now show one possible implementation of the PMCA that, on input $G_{10,k}(\beta)$, returns a Hamiltonian cycle of length $3(k - 1)^2$.

![Figure 4.2: A Hamiltonian cycle of length $2k + 2\beta^2 + 7\beta + 6$ in $G_{10,k}(\beta)$.](image)

4.1 Minimum Spanning Tree

The PMCA computes the minimum spanning tree with all edges of length $1/k$, the edges $\{v_{5,i}, v_{6,i}\}$, for $1 \leq i \leq k$, and the edges $\{v_{9,i}, v_{2,i+1}\}$, for $1 \leq i \leq k - 1$, as shown in Figure 4.3, resulting in the set of odd vertices

$$ U = \{v_{1,i}, v_{3,i}, v_{4,i}, v_{6,i}, v_{7,i}, v_{8,i}, v_{10,i} | 1 \leq i \leq k\} \cup \{v_{2,i} | 2 \leq i \leq k\} \cup \{v_{9,k}\}. $$

It is easy to see that $T$ is indeed a minimum spanning tree. Every edge in the graph has length at least $1/k$, therefore the edges $\{v_{1,i}, v_{2,i}\}$, $\{v_{2,i}, v_{5,i}\}$, $\{v_{4,i}, v_{5,i}\}$, $\{v_{5,i}, v_{9,i}\}$, $\{v_{8,i}, v_{9,i}\}$, $\{v_{9,i}, v_{10,i}\}$, for $1 \leq i \leq k$, form minimum spanning trees for the respective upper subclusters. On the other hand, the
edges \(\{v_{3,i}, v_{6,i}\}, \{v_{6,i}, v_{7,i}\}\), for \(1 \leq i \leq k\), form minimum spanning trees for the respective lower subclusters. All we need to do is add an edge for every component to construct a minimum spanning tree for the whole graph. All edges available for this have length at least 1, so we can just take the edges \(\{v_{5,i}, v_{6,i}\}, \text{ for } 1 \leq i \leq k\), and \(\{v_{9,i}, v_{2,i+1}\}, \text{ for } 1 \leq i \leq k - 1\).

\[
\begin{align*}
\text{Figure 4.3:} & \quad \text{A minimum spanning tree in } G_{10,k}(\beta). \text{ The odd vertices are circled.}
\end{align*}
\]

### 4.2 Minimum Path Matching

The PMCA computes the minimum path matching

\[
\Pi = \{(v_{1,i}, v_{2,i}), (v_{4,i}, v_{5,i}, v_{6,i}, v_{3,i}), (v_{6,i}, v_{7,i}),
(v_{8,i}, v_{9,i}, v_{10,i}) \mid 2 \leq i \leq k - 1\} \cup
\{(v_{4,1}, v_{5,1}, v_{6,1}, v_{3,1}), (v_{6,1}, v_{7,1}), (v_{8,1}, v_{9,1}, v_{10,1}), (v_{1,1}, v_{7,1}),
(v_{1,k}, v_{2,k}), (v_{3,k}, v_{6,k}), (v_{4,k}, v_{5,k}, v_{9,k}), (v_{8,k}, v_{9,k}, v_{10,k})\}
\]

as shown in Figure 4.4.

**Theorem 4.3.** \(\Pi\) is a minimum path matching for \(U\).

In order to prove this result, we need three lemmas.

**Lemma 4.4.** In a minimum path matching, exactly one vertex per lower subcluster is matched with a vertex that is not in this lower subcluster.

**Proof.** The “at least” part holds because \(|LSC_i \cap U|\) is odd.

Assume for contradiction that the “at most” part is wrong. Then, there is a minimum path matching that contains paths \((v_{3,i}, \ldots, u), (v_{6,i}, \ldots, v)\),
(v_7,i, ..., w) for some vertices u, v, w \notin LSC_i. We replace them with shortest paths (v_3,i, ..., u), (v_6,i, v_7,i), (v, ..., w) to obtain a strictly shorter path matching for sufficiently large k.

**Case 1:** v, w are USC vertices. Then, \( c(v_6,i, ..., v) + c(v_7,i, ..., w) \geq 2 + 3/k \). The distance from v to w is at most \( 1 + 4/k \). Therefore, \( c(v_6,i, v_7,i) + c(v, ..., w) \leq 1 + 5/k \).

**Case 2:** v is a USC vertex, w is an LSC vertex. Then, \( c(v_6,i, ..., v) + c(v_7,i, ..., w) \geq 3 + 5/k \). The distance from v to w is at most \( 1 + 8/k \). Therefore, \( c(v_6,i, v_7,i) + c(v, ..., w) \leq 1 + 9/k \). The inverse case works analogously.

**Case 3:** v, w are LSC vertices. Then, \( c(v_6,i, ..., v) + c(v_7,i, ..., w) \geq 4 + 9/k \). The distance from v to w is at most \( 2 + 7/k \). Therefore, \( c(v_6,i, v_7,i) + c(v, ..., w) \leq 2 + 8/k \).

**Lemma 4.5.** In a minimum path matching, at most one of the vertices \( v_{8,i} \) and \( v_{10,i} \) per cluster is matched with an LSC vertex.

**Proof.** Assume for contradiction that there are paths \((v_{8,i}, ..., v), (v_{10,i}, ..., w)\) for some LSC vertices v, w. We replace these two paths with \((v_{8,i}, v_{9,i}, v_{10,i})\) and a shortest path \((v, ..., w)\) to obtain a strictly shorter path matching for sufficiently large k. We know from Lemma 4.4 that v and w are not in the same lower subcluster. In particular, at least one of them is not in cluster \( C_i \), which gives us the desired bounds.

**Case 1:** \( v = v_{3,m}, w = v_{3,n} \).

---

1. Remember that any two upper subclusters are connected by an edge of length 1.
2. Remember that any upper and lower subcluster not belonging to the same cluster are connected by an edge of length \( 1 + 2/k \).
3. In this case, the shortest path from v to w is via some vertex \( a_i \) and contains one edge of length \( 1 + 2/k \) and one edge of length 1.
Chapter 4. On the Approximation Ratio of the PMCA

Let us first consider the case where \( m = i \). Then the shortest path from \( v_{8,i} \) to \( v \) is of the form

\[(v_{8,i}, v_{9,i}, v_{5,i}, v_{6,i}, v_{3,i})\]

and has length \( 1 + 3/k \). The shortest path from \( v_{10,i} \) to \( w \) is of the form

\[(v_{10,i}, v_{9,i}, v_{5,i}, v_{2,i}, v_{1,i}, v_{7,m}, v_{6,m}, v_{3,m})\]

and has length \( 1 + 8/k \). The inverse case works analogously.

If \( m \neq i \) and \( n \neq i \), both paths have the latter form and thus length \( 1 + 8/k \). Therefore, the minimum length of the two shortest paths \((v_{8,i}, \ldots, v_{3,m})\), \((v_{10,i}, \ldots, v_{3,n})\) is \( 2 + 11/k \). But we can replace them with the path

\[(v_{3,m}, v_{6,m}, v_{5,m}, v_{2,m}, v_{1,m}, v_{7,n}, v_{6,n}, v_{3,n})\]

of length \( 2 + 7/k \) and \((v_{8,i}, v_{9,i}, v_{10,i})\) of length \( 2/k \), thus obtaining a shorter minimum path matching.

**Case 2:** \( v = v_{3,m}, w = v_{6,n} \). Then, \( c(v_{8,i}, \ldots, v) + c(v_{10,i}, \ldots, w) \geq 2 + 10/k \) but \( c(v_{8,i}, v_{9,i}, v_{10,i}) + c(v, \ldots, w) \leq 2 + 8/k \). The inverse case works analogously.

**Case 3:** \( v = v_{3,m}, w = v_{7,n} \). Then, \( c(v_{8,i}, \ldots, v) + c(v_{10,i}, \ldots, w) \geq 2 + 9/k \) but \( c(v_{8,i}, v_{9,i}, v_{10,i}) + c(v, \ldots, w) \leq 2 + 7/k \). The inverse case works analogously.

**Case 4:** \( v = v_{6,m}, w = v_{6,n} \). Then, \( c(v_{8,i}, \ldots, v) + c(v_{10,i}, \ldots, w) \geq 2 + 9/k \) but \( c(v_{8,i}, v_{9,i}, v_{10,i}) + c(v, \ldots, w) \leq 2 + 7/k \).

**Case 5:** \( v = v_{6,m}, w = v_{7,n} \). Then, \( c(v_{8,i}, \ldots, v) + c(v_{10,i}, \ldots, w) \geq 2 + 8/k \) but \( c(v_{8,i}, v_{9,i}, v_{10,i}) + c(v, \ldots, w) \leq 2 + 6/k \). The inverse case works analogously.

**Case 6:** \( v = v_{7,m}, w = v_{7,n} \). Then, \( c(v_{8,i}, \ldots, v) + c(v_{10,i}, \ldots, w) \geq 2 + 9/k \) but \( c(v_{8,i}, v_{9,i}, v_{10,i}) + c(v, \ldots, w) \leq 2 + 6/k \).

**Lemma 4.6.** Every minimum path matching can be transformed into a minimum path matching in which (i) no \( v_{8,i} \) is matched with an LSC vertex and (ii) \( v_{9,k} \) is matched neither with \( v_{8,k} \) nor with \( v_{10,k} \).

**Proof.**

(i) If the matching contains no such paths, we go to step (ii). Otherwise, let \((v_{8,i}, \ldots, l)\) for some LSC vertex \( l \) be one such path. We have to show a transformation that does not increase the size of the matching. This is easy. We know from Lemma 4.5 that there exists a path \((v_{10,i}, \ldots, u)\) for some USC vertex \( u \). We can assume w.l.o.g. that these two paths have the form \((v_{8,i}, v_{9,i}, \ldots, l)\) and \((v_{10,i}, v_{9,i}, \ldots, u)\), respectively. So we can replace them with shortest paths \((v_{8,i}, v_{9,i}, \ldots, u)\) and \((v_{10,i}, v_{9,i}, \ldots, l)\), i.e., we just “flipped” \( v_{8,i} \) and \( v_{10,i} \).
4.2. Minimum Path Matching

(ii) If the modified matching contains no path violating the second condition, we are done. Otherwise it contains w.l.o.g. the path \((v_{8,k}, v_{9,k})\) because if it contains \((v_{9,k}, v_{10,k})\), we can just “flip” \(v_{8,k}\) and \(v_{10,k}\) as above. We replace \((v_{8,k}, v_{9,k})\) and \((v_{10,k}, \ldots, v)\) with \((v_{8,k}, v_{9,k}, v_{10,k})\) and a shortest path \((v_{9,k}, \ldots, v)\) As above, we know from the construction of our graph that these two paths have at most the same length. This transformation does obviously not violate condition (i).

Now we are finally able to prove Theorem 4.3.

Proof. Let us first compute the length of \(\Pi\). It contains \(k - 1\) paths \((v_{4,i}, v_{5,i}, v_{6,i}, v_{3,i})\) and \((v_{1,j}, v_{2,j})\), respectively \((1 \leq i \leq k - 1, 2 \leq j \leq k)\), of length \(1 + 2/k\) and \(1/k\), respectively, summing up to \((k - 1) \cdot (1 + 2/k + 1/k) = k + 2 - 3/k\). Furthermore, it contains \(k\) paths \((v_{8,i}, v_{9,i}, v_{10,i})\) of length \(2/k\) as well as \(k\) paths of length \(1/k\) connecting two LSC vertices, summing up to 3. The remaining two paths are \((v_{4,k}, v_{5,k}, v_{9,k})\) and \((v_{1,1}, v_{7,k})\) of length \(2/k\) and \(1 + 2/k\), respectively. Thus the length of \(\Pi\) is \(k + 2 - 3/k + 2/k + 1 + 2/k = k + 6 + 1/k\).

Therefore, assume for contradiction that there is a minimum path matching \(\Pi'\) of length less than \(k + 6 + 1/k\) that, w.l.o.g., satisfies conditions (i) and (ii) of Lemma 4.6. We now compute the length of \(\Pi'\).

First consider all paths having an LSC vertex as an endpoint. We know from Lemma 4.4 that, for every \(LSC_i\), this means two paths \((l_1, \ldots, l_2)\) and \((l_3, \ldots, v)\), where \(l_1, l_2, l_3\) denote the three vertices in \(LSC_i\) and \(v\) is some vertex outside \(LSC_i\). It is possible that \(v \in LSC_j\). Let therefore \(c\) be the number of paths matching two LSC vertices from different clusters. No matter what vertex we choose as \(l_3\), the two paths always have length at least \(1 + 3/k\). On the other hand, a path connecting two LSC vertices from different clusters \(C_i\) and \(C_j\) has a length of at least \(2 + 4/k\), but we still have to consider the paths inside \(LSC_i\) and \(LSC_j\), respectively. Summing up, the minimum length of all paths having one or possibly two LSC vertices as endpoints amounts to

\[
k \cdot 1/k + (k - 2c) \cdot (1 + 2/k) + c \cdot (2 + 4/k) = k + 3.
\]

Now we compute the length of all paths having some vertex \(v_{8,i}\) as an endpoint. A path of the form \((v_{8,i}, \ldots, v)\), for some vertex \(v \neq v_{8,j}\), has minimum length \(2/k\). But we again have to consider the possibility of paths \((v_{8,i}, \ldots, v_{8,j})\). Such a path has minimum length \(1 + 4/k\). Let \(d\) be the number of such paths. Then, the minimum length of all paths having one or possibly two vertices \(v_{8,j}, v_{8,j}\) as endpoints amounts to

\[
(k - 2d) \cdot 2/k + d \cdot (1 + 4/k) = 2 + d \geq 2.
\]
Observe that the two path sets considered above are disjoint because of condition (i) of Lemma 4.6.

Now we consider all remaining paths. Because the two path sets above contain together at most $3k$ paths, we are left with at least $k$ paths. Every path has length at least $1/k$. But there is more to it. All possible paths of length $1/k$ have the form $(v_{3,i}, v_{6,i}, v_{7,i})$, for $1 \leq i \leq k$, or $(v_{1,j}, v_{2,j})$, for $2 \leq j \leq k$, or $(v_{8,k}, v_{9,k})$ or $(v_{9,k}, v_{10,k})$. But we already considered above all paths of the first two forms, and the paths of the last two forms cannot occur in $\Pi'$ because of condition (ii) of Lemma 4.6. Therefore only the paths $(v_{1,j}, v_{2,j})$, for $2 \leq j \leq k$, i.e., only $k-1$ paths of length $1/k$ are left. Since every other path has length at least $2/k$, the minimum length of all remaining paths amounts to $1+1/k$.

We have shown that $\Pi'$ has minimum length

$$k + 3 + 2 + 1 + 1/k = k + 6 + 1/k,$$

which is a contradiction to our initial assumption. \qed

### 4.3 Conflict Resolution in the Minimum Path Matching

The goal of this step is to resolve all conflicts in the minimum path matching $\Pi$ to obtain a vertex-disjoint path matching $\Pi'$. Procedure 3.3 does this for every connected component of $\Pi$ separately. The paths

$$\{(v_{1,1}, v_{7,k})\} \cup \{(v_{1,i}, v_{2,i}) \mid 2 \leq i \leq k\} \cup \{(v_{8,i}, v_{9,i}, v_{10,i}) \mid 1 \leq i \leq k-1\},$$

contain no conflicts and hence there is nothing to do.

Let us therefore look at the problematic paths of $\Pi$, i.e.,

$$\{(v_{4,i}, v_{5,i}, v_{6,i}, v_{3,i}), (v_{6,i}, v_{7,i}) \mid 1 \leq i \leq k-1\} \cup\{(v_{4,k}, v_{5,k}, v_{9,k}), (v_{8,k}, v_{9,k}, v_{10,k})\}.$$ 

For each component in the first set, Procedure 3.3 may choose the path $(v_{4,i}, v_{5,i}, v_{6,i}, v_{3,i})$ as $p$ and thus bypass $v_{6,i}$ in this path. For the second set, Procedure 3.3 may choose the path $(v_{8,i}, v_{9,i}, v_{10,i})$ as $p$ and thus bypass $v_{9,i}$ in this path.

In this step, the PMCA thus computes the vertex-disjoint path matching

$$\Pi' := \{(v_{1,1}, v_{2,i}), (v_{4,i}, v_{5,i}, v_{3,i}), (v_{6,i}, v_{7,i}), (v_{8,i}, v_{9,i}, v_{10,i}) \mid 2 \leq i \leq k-1\} \cup \{(v_{4,1}, v_{5,1}, v_{3,1}), (v_{6,1}, v_{7,1}), (v_{8,1}, v_{9,1}, v_{10,1}), (v_{1,1}, v_{7,k}), (v_{1,k}, v_{2,k}), (v_{3,k}, v_{6,k}), (v_{4,k}, v_{5,k}, v_{9,k}), (v_{8,k}, v_{10,k})\}$$

as shown in Figure 4.5.
4.4 Eulerian Cycle

Alternating between paths from $T$ and paths from $\Pi'$, the PMCA now computes the Eulerian cycle

$$E = (p_1, q_2, p_2, q_3, p_3, q_4, p_4, q_5, \ldots, p_{4k-4}, q_{4k-3}, p_{4k-3}, q_{4k-2}, p_{4k-2}, q_{4k-1}, p_{4k-1}, q_{4k}, p_{4k}, q_1)$$

$$= (v_1, v_2, v_3, v_4, v_5, v_6, v_7, v_8, v_9, v_{10}, v_9, v_8, v_7, v_6, v_5, v_4, v_3, v_2, v_1)$$

as shown in Figure 4.6. For clarity, the paths of $\Pi'$ are dashed.
4.5 Conflict Resolution in the Minimum Spanning Tree

The goal of this section is to introduce some bypasses such that every vertex is adjacent to at most three edges from \( T \). The problematic vertices are all \( v_{5,i} \) and all \( v_{9,i} \) except \( v_{9,k} \). Let \( r := v_{1,1} \). Then, the PMCA bypasses the vertices \( v_{5,i} \) between \( v_{6,i} \) and \( v_{9,i} \), for \( 1 \leq i \leq k - 1 \), the vertex \( v_{5,k} \) between \( v_{9,k} \) and \( v_{6,k} \), and the vertices \( v_{9,i} \) between \( v_{8,i} \) and \( v_{2,i+1} \), for \( 1 \leq i \leq k - 1 \), to obtain the modified Eulerian cycle

\[
E' = (v_{1,1}, v_{2,1}, v_{5,1}, v_{4,1}, v_{5,1}, v_{3,1}, v_{6,1}, v_{7,1}, v_{6,1}, v_{9,1}, v_{10,1}, v_{9,1}, v_{8,1},
\]

\[
v_{2,2}, v_{1,2}, \ldots, v_{5,k}, v_{9,k}, v_{8,k}, v_{10,k}, v_{9,k}, v_{6,k}, v_{3,k}, v_{6,k}, v_{7,k}, v_{1,1})
\]

as shown in Figure 4.7.

![Figure 4.7: The modified Eulerian cycle \( E' \). The paths of \( \Pi' \) are dashed.](image)

4.6 Conflict Resolution in the Modified Eulerian Cycle

The goal of the last step is to bypass vertices such that every vertex has degree 2, i.e., we have a Hamiltonian cycle. The problematic vertices are thus all vertices \( v_{2,i} \), except \( v_{2,1} \) and all vertices \( v_{5,i}, v_{6,i}, v_{9,i} \).

The PMCA obtains the final Hamiltonian cycle

\[
H = (v_{1,1}, v_{2,1}, v_{5,1}, v_{4,1}, v_{3,1}, v_{6,1}, v_{7,1}, v_{9,1}, v_{10,1}, v_{8,1}, v_{1,2}, \ldots,
\]

\[
v_{8,k-1}, v_{1,k}, v_{2,k}, v_{5,k}, v_{4,k}, v_{9,k}, v_{8,k}, v_{10,k}, v_{6,k}, v_{3,k}, v_{7,k}, v_{1,1})
\]

as shown in Figure 4.8 by doing the following:
• it bypasses \( v_{2,i} \) between \( v_{8,i-1} \) and \( v_{1,i} \), for \( 2 \leq i \leq k \);

• it bypasses \( v_{5,i} \) between \( v_{4,i} \) and \( v_{3,i} \), for \( 1 \leq i \leq k-1 \), and \( v_{5,k} \) between \( v_{4,k} \) and \( v_{9,k} \);

• it bypasses \( v_{6,i} \) between \( v_{7,i} \) and \( v_{9,i} \), for \( 1 \leq i \leq k-1 \), and \( v_{6,k} \) between \( v_{3,k} \) and \( v_{7,k} \);

• it bypasses \( v_{9,i} \) between \( v_{10,i} \) and \( v_{8,i} \), for \( 1 \leq i \leq k-1 \), and \( v_{9,k} \) between \( v_{10,k} \) and \( v_{6,k} \).

\[ v_{1,1}, v_{4,1}, v_{8,1}, v_{1,2}, v_{4,2}, v_{8,2} \]

\[ v_{2,1}, v_{5,1}, v_{9,1}, v_{2,2}, v_{5,2}, v_{9,2} \]

\[ v_{3,1}, v_{7,1}, v_{10,1}, v_{3,2}, v_{7,2}, v_{10,2} \]

\[ \ldots v_{2,k}, v_{5,k}, v_{9,k}, v_{6,k}, v_{10,k} \]

\[ v_{7,k} \]

Figure 4.8: The Hamiltonian cycle \( H \).

Considering only the edges \( \{v_{4,i}, v_{3,i}\}, \{v_{7,i}, v_{9,i}\}, \{v_{8,i}, v_{1,i+1}\} \), for \( 1 \leq i \leq k-1 \), we obtain \( \text{cost}(H) \geq 3(k-1)\beta^2 \). We have thus shown that, for every \( \beta \geq 1 \) and arbitrarily small \( \varepsilon > 0 \), there exists an implementation \( I \) of the PMCA such that

\[
\frac{\text{cost}(I(G_{10,k}(\beta)))}{\text{Opt}_{\Delta_x,\text{TSP}}(G_{10,k}(\beta))} \geq \frac{3(k-1)\beta^2}{2k + 2\beta^2 + 7\beta + 6} \geq \frac{3}{2} \beta^2 - \varepsilon,
\]

for sufficiently large \( k \), i.e., we have shown that the upper bound of \( \frac{3}{2} \beta^2 \) on the approximation ratio of the PMCA is tight.
Chapter 4. On the Approximation Ratio of the PMCA
Chapter 5

The PMCA for the Hamiltonian Path Problem

In this chapter, we present a modification of the PMCA for the Hamiltonian path problem in near-metric graphs with zero, one, or two fixed endpoints ($\Delta_{\beta}$-HPP$_0$, $\Delta_{\beta}$-HPP$_1$, and $\Delta_{\beta}$-HPP$_2$, respectively). This algorithm is due to Forlizzi et al. [FHP06], who combined Hoogeveen’s modification of the Christofides algorithm for the Hamiltonian path problem [Hoo91] with the PMCA and showed that the approximation ratio of the resulting algorithm is bounded from above by $\frac{5}{2}\beta^2$ for the $\Delta_{\beta}$-HPP$_0$ and the $\Delta_{\beta}$-HPP$_1$, and by $\frac{5}{3}\beta^2$ for the $\Delta_{\beta}$-HPP$_2$.

Definition 5.1. The $\beta$-metric Hamiltonian path problem without fixed endpoints ($\Delta_{\beta}$-HPP$_0$) is the following optimization problem.

**Input:** A complete weighted graph $G = (V, E, c)$ satisfying the $\beta$-triangle inequality.

**Constraints:** $M(G, c) = \{(v_1, \ldots, v_n) \mid (v_1, \ldots, v_n) \text{ is a permutation of } V\}$ is the set of all Hamiltonian paths in $G$.

**Costs:** The cost of a Hamiltonian path $P = (v_1, \ldots, v_n)$ is $c(P) = \sum_{i=1}^{n-1} c(v_i, v_{i+1})$.

**Goal:** Minimum.

Definition 5.2. The $\beta$-metric Hamiltonian path problem with one fixed endpoint ($\Delta_{\beta}$-HPP$_1$) is the following optimization problem.

**Input:** A complete weighted graph $G = (V, E, c)$ satisfying the $\beta$-triangle inequality and a distinct vertex $v \in V$. 

\[\text{...}\]
Constraints: \( M(G, c) = \{(v, v_1, \ldots, v_{n-1}) \mid (v_1, \ldots, v_{n-1}) \text{ is a permutation of } V - \{v\}\} \) is the set of all Hamiltonian paths in \( G \) having \( v \) as an endpoint.

Costs: The cost of a Hamiltonian path \( P = (v, v_1, \ldots, v_{n-1}) \) is \( c(P) = c(v, v_1) + \sum_{i=1}^{n-2} c(v_i, v_{i+1}) \).

Goal: Minimum.

Definition 5.3. The \( \beta \)-metric Hamiltonian path problem with two fixed endpoints (\( \Delta_\beta\text{-HPP}_2 \)) is the following optimization problem.

Input: A complete weighted graph \( G = (V, E, c) \) satisfying the \( \beta \)-triangle inequality and two distinct vertices \( v, w \in V \).

Constraints: \( M(G, c) = \{(v, v_1, \ldots, v_{n-2}, w) \mid (v_1, \ldots, v_{n-2}) \text{ is a permutation of } V - \{v, w\}\} \) is the set of all Hamiltonian paths in \( G \) having \( v \) and \( w \) as endpoints.

Costs: The cost of a Hamiltonian path \( P = (v, v_1, \ldots, v_{n-2}, w) \) is \( c(v, v_1) + \sum_{i=1}^{n-3} c(v_i, v_{i+1}) + c(v_{n-2}, w) \).

Goal: Minimum.

5.1 The Algorithm

The algorithm is shown in Procedure 5.1. Observe that it is quite similar to the PMCA (Procedure 3.2). The following changes are necessary. First, \( U \) is computed differently. It no longer contains all odd vertices in \( T \), but only those that are not prespecified endpoints plus those endpoints with even degree. Step 2 constructs a path matching \( \Pi \) such that the multigraph formed by \( T \) and \( \Pi \) contains exactly two odd vertices. Steps 3, 5 and 6 remain the same—but only in this high-level description. The actual implementations of these steps have to deal with quite a few intricate technical details, mainly concerning the prespecified endpoints. In step 4, we of course need to make sure that the constructed Eulerian path starts and ends in the prespecified endpoints.

The main difficulty in all steps arises from the fact that we cannot bypass the prespecified endpoints.

The implementation of step 1 is well-known [CLRS09]. The implementation of step 4 is also well-known [GP05], but we need an Eulerian path with a specific structure for steps 5 and 6. Therefore, we use a customized algorithm.
that constructs such an Eulerian path. The following sections present the implementations of steps 2 to 6 from [FHPS06].

**Procedure 5.1** PMCA for the $\Delta_\beta$-HPP$_l$ (PMCA-HPP$_l$)

**Input:** A complete $\beta$-metric graph $G = (V,E)$, for some $\beta \geq 1$, and a set $A \subseteq V$ of size $l$.

1: Find a minimum spanning tree $T$ in $G$. Let $U$ be the vertices in $V - A$ having odd degree in $T$ plus the vertices in $A$ having even degree in $T$.
2: Construct a minimum path matching $\Pi$ for $U$. If necessary, remove an edge from $T$ such that the multigraph formed by $T$ and $\Pi$ has two odd vertices $w$ and $z$.
3: Resolve conflicts in $\Pi$ in order to obtain a vertex-disjoint path matching $\Pi'$ in which $z$ does not occur as an inner vertex.
4: Construct an Eulerian path $P = (p_1, q_1, p_2, q_2, \ldots)$ on $T$ and $\Pi'$ from $w$ to $z$ such that $p_1, p_2, \ldots$ are paths in $T$ and $q_1, q_2, \ldots$ are paths in $\Pi'$.
5: Transform $p_1, p_2, \ldots$ into $p'_1, p'_2, \ldots$ such that the forest $T_f$ formed by $p'_1, p'_2, \ldots$ has degree at most 3, $w$ and $z$ are the endpoints of $P'' := (p'_1, q_1, p'_2, q_2, \ldots)$ and $z$ is not a conflict in $P''$.
6: Resolve all remaining conflicts in $P''$ to obtain a Hamiltonian path $P''$ from $w$ to $z$.

**Output:** $P''$.

### 5.1.1 Construction of a Minimum Path Matching

To construct a minimum path matching, we use a simple trick. As in the PMCA, we compute a complete graph $G'$ over $U$ where the length of an edge $\{u, v\}$ corresponds to the length of a shortest path between $u$ and $v$ in $G$. Then, we add $2 - l$ dummy vertices to $G'$. All edges adjacent to those dummy vertices have length 0, except for the edge between them (if $l = 0$), which has length $\infty$. We compute a minimum matching $M$ in $G'$ and remove the edges adjacent to dummy vertices to obtain $M'$. Finally, we map $M'$ back to $G$ by connecting two vertices in $G$ via a shortest path if they are matched in $M'$.

It is easy to see that the multigraph formed by $T$ and the obtained path matching $\Pi$ contains two or zero odd vertices. It contains zero odd vertices if there is a single prespecified endpoint $s$ that has even degree in $T$ and is not an endpoint of a path in $\Pi$. (That is, $s$ was connected to the dummy vertex in $G'$.) In this case, we remove an arbitrary edge incident to $s$ in $T$.

Let $w$ and $z$ be the two odd vertices in the resulting multigraph.

**Lemma 5.4.** Any prespecified endpoint lies in $\{w, z\}$.
Chapter 5. The PMCA for the Hamiltonian Path Problem

Proof. The statement is trivially true if \( l = 0 \).

If \( l = 1 \), let \( s \) be the prespecified endpoint. If \( s \) has odd degree in \( T \), we are fine, since then it also has odd degree in the multigraph formed by \( T \) and \( \Pi \). If \( s \) has even degree, there are two possibilities. Either \( s \) is not matched with the unique dummy vertex in \( G' \) and thus is an endpoint of a path in \( \Pi \). Then, it has odd degree afterwards. Or \( s \) is matched with the dummy vertex and is thus not an endpoint of a path in \( \Pi \). In this case, the rule above applies and we remove an arbitrary edge adjacent to \( s \) in \( T \), thus ensuring that \( s \) is indeed also of odd degree in the multigraph formed by \( T \) and \( \Pi \).

If \( l = 2 \), let \( s \) be a prespecified endpoint. If \( s \) has even degree in \( T \), then, since there are no dummy vertices in \( G' \), \( s \) is an endpoint of a path in \( \Pi \) and thus has odd degree in the end. If \( s \) has odd degree, nothing changes. \( \square \)

5.1.2 Conflict Resolution in the Minimum Path Matching

Now we transform the cycle-free, edge-disjoint path matching \( \Pi \) into a vertex-disjoint path matching \( \Pi' \). We do this for every tree in \( \Pi \) separately. The implementation of this step is different to the implementation in the PMCA, because we need to ensure that the vertex \( z \) does not occur as an internal vertex on any path in \( \Pi' \).

**Procedure 5.2 Decompose-Tree**

**Input:** A path matching \( \Pi_T \) forming a tree and a vertex \( x \) that occurs in some path \( q \) in \( \Pi_T \).

1: while \( q \) do
2: let \( v \) be a conflict of maximum distance from \( x \) in \( q \) such that w.l.o.g. \( q = (a, \ldots, x, \ldots, v, b, \ldots, c) \) where \( b, \ldots, c \) are no conflicts.
3: let \( q_1, \ldots, q_k \) be the paths forming the tree containing \( v \) in \( \Pi_T - \{q\} \)
4: \( \{q_1, \ldots, q_k\} := \text{Decompose-Tree}(\{q_1, \ldots, q_k\}, v) \) such that w.l.o.g. \( v \in q_1 \)
5: if \( v \) is internal to \( q_1 \) then
6: bypass \( v \) in \( q_1 \)
7: else
8: assume w.l.o.g. \( q_1 = (d, \ldots, e, v) \)
9: transform \( q \) and \( q_1 \) into \( (a, \ldots, x, \ldots, v) \) and \( (d, \ldots, e, b, \ldots, c) \)
10: end
11: end

**Output:** A vertex-disjoint path matching \( \Pi'_T \) in which \( x \) still occurs.

Let us first look at Procedure 5.2. It takes as input a path matching...
$\Pi_T$ forming a tree and a vertex $x$ that occurs in some path of $\Pi_T$. Then, it transforms $\Pi_T$ into a vertex-disjoint path matching $\Pi'_T$ in which $x$ still occurs in some path. It does this in the following way.

It picks some path $q$ in which $x$ occurs and then searches for the conflict $v$ in $q$ farthest away from $x$. After that, it (recursively) transforms the path matching tree $\Pi_v$ containing $v$ after the removal of $q$ into a vertex-disjoint path matching $\Pi'_v$. The recursion guarantees that $v$ occurs in some (unique) path $q_1$ in $\Pi'_v$. If $v$ is internal to $q_1$, we can simply bypass it in $q$. Otherwise, we have to do the transformation as shown in Figure 3.3.

**Procedure 5.3**

**Input:** An edge-disjoint, cycle-free path matching $\Pi$ for $U$ in $G$.

1. for all trees $\Pi_T$ in $\Pi$ do
2. if $\Pi_T$ contains $z$ then
3. $\Pi'_T := \text{Decompose-Tree}(\Pi_T, z)$
4. if $z$ is internal to some path $q$ in $\Pi'_T$ then
5. bypass $z$ in $q$
6. end
7. else
8. let $x$ be an arbitrary vertex in $\Pi_T$
9. $\Pi'_T := \text{Decompose-Tree}(\Pi_T, x)$
10. end
11. end

**Output:** A vertex-disjoint path matching $\Pi'$ in which $z$ is no internal vertex.

Procedure 5.3 not only resolves all conflicts in $\Pi$, but also removes all internal occurrences of $z$. It does this by successively transforming all path matching trees $\Pi_T$ into vertex-disjoint path matchings $\Pi'_T$ using Procedure 5.2. For each $\Pi_T$, there are two possibilities. If $\Pi_T$ does not contain $z$ we do not have to take any precautions, i.e., we can take an arbitrary vertex. ($\Pi'_T$ does not contain $z$ anyway.) If $\Pi_T$ contains $z$, Procedure 5.2 ensures that $z$ also occurs in the obtained vertex-disjoint path matching $\Pi'_T$. If this occurrence is internal to some (unique) path $q$ in $\Pi'_T$, then we bypass $z$ in $q$.

**Lemma 5.5** [FHPS06]. Procedure 5.3 returns a vertex-disjoint path matching that contains no internal occurrences of $z$. Moreover, every path in $\Pi'$ has at most one bypass and every bypass is of size 2.

---

1This step is similar to line 8 of Procedure 3.3 where we searched for the first or last conflict in a path.
5.1.3 Construction of an Eulerian Path

Now we construct an Eulerian path from \( w \) to \( z \).\(^2\) We need to distinguish two cases. Let \( y \) be the unique neighbor of \( w \) towards \( z \) in the minimum spanning tree constructed in step 1.

**Case 1:** There exists a path \( p = (z, \ldots, w) \) in \( \Pi' \). Then, we first construct an Eulerian cycle \( E = (w, u_1, \ldots, u_{h-1}, w) \) on \( T \) and \( \Pi' - \{p\} \). We can now concatenate \( p \) and \( E \) to obtain the Eulerian path \( (z, \ldots, w, u_1, \ldots, u_{h-1}, w) \).

**Case 2:** There exists no such path. Then, we apply Procedure 5.4, which essentially does the following. It looks if there are (unique) paths \( p = (z, \ldots, u) \) and \( q = (u', \ldots, w) \), and if so, searches for an Eulerian path \( P \) from \( u \) to \( u' \) and returns the concatenation of \( p, P, \) and \( q \).

**Procedure 5.4**

**Input:** A minimum spanning tree \( T \), a vertex-disjoint path matching \( \Pi' \) and two distinct vertices \( w \) and \( z \).

1. \( v := z, P := \Pi' \)
2. **if** there is a path \( p = (z, \ldots, u) \) in \( \Pi' \) **then**
3. \( v := u, P := \Pi' - \{p\} \)
4. **end**
5. **if** there is a path \( q = (u', \ldots, w) \) in \( \Pi' \) **then**
6. construct an Eulerian path \( P \) from \( v \) to \( u' \) in \( T \) and \( P - \{q\} \)
7. append \( q \) to \( P \)
8. **else**
9. let \( y \) be the unique neighbor of \( w \) towards \( z \)
10. **if** \( T \) still contains the edge \( \{y, w\} \) **then**
11. construct an Eulerian path \( P \) from \( v \) to \( y \) in \( T - \{y, w\} \) and \( P \)
12. append \( \{y, w\} \) to \( P \)
13. **else**
14. (We might have deleted \( \{y, w\} \) in step 2 of the PMCA-HPP.)
15. construct an Eulerian path \( P \) from \( v \) to \( w \) in \( T \) and \( P \)
16. **end**
17. **end**
18. **if** \( v \neq z \) **then**
19. append \( p \) at the beginning of \( P \)
20. **end**

**Output:** The Eulerian path \( P \) from \( z \) to \( w \).

**Lemma 5.6** [FHPS06]. Procedure 5.4 returns an Eulerian path satisfying the

\(^2\)Remember that any prespecified endpoint lies in \( \{w, z\} \), therefore this will always yield a valid solution.
5.1. The Algorithm

following properties.

- Every vertex except $w$ occurs at most once as an endpoint of a path in $T$.
- $z$ occurs as an endpoint of either a path in $T$ or a path in $\Pi'$.
- If the occurrence of $w$ that is an endpoint of $P$ is an endpoint of a path $p$ in $T$, then each occurrence of $w$ internal to $P$ that is contained in a path $q$ in $T$ is the vertex in $q$ with minimum distance from $z$ in $T$. 

5.1.4 Conflict Resolution in the Minimum Spanning Tree

Procedure 5.5 resolves conflicts in the minimum spanning tree similarly to the PMCA, with the only difference that we do not arbitrarily choose a root $r$, but we set $r := z$. Note that this avoids the problem from the PMCA—no vertex is dropped completely.

**Procedure 5.5**

**Input:** The paths $p_1, p_2, \ldots$ forming $T$ computed in step 4 of the PMCA-HPP$_t$.

1: for all paths $p_i$ in $T$ do
2: let $v_j$ be the vertex in $p_i$ of minimal distance to $z$ in $T$
3: if $v_j$ is internal to $p_i$ then
4: bypass $v_j$ in $p_i$
5: end
6: call the resulting path $p'_i$
7: end

**Output:** The paths $p'_1, p'_2, \ldots$ forming a forest $T_f$.

**Lemma 5.7** [BHK+02]. The resulting Eulerian path $P' := (p'_1, q_1, p'_2, q_2, \ldots)$ has $w$ and $z$ as its endpoints. Every vertex occurs once or twice in $P'$, and $z$ occurs exactly once. Moreover, every path in $T_f$ has at most one bypass, and every bypass has size 2. 

5.1.5 Conflict Resolution in the Modified Eulerian Path

Procedure 5.6 implements this last step in almost the same way as in the PMCA, with the only difference that we do not start with an arbitrary vertex,
but with $w$.³

**Procedure 5.6**

**Input:** The Eulerian path $P'$ computed in step 5 of the PMCA-HPP.

1. if $w$ is a conflict in $P'$ then
2. let $u$ be the occurrence of $w$ that is not an endpoint of $P'$
3. else
4. let $u$ be an arbitrary conflict in $P'$
5. end
6. bypass $u$
7. while there are conflicts remaining do
8. if occurrence $u$ has at least one unresolved conflict as neighbor then
9. let $v$ be one of them, chosen by Rule 5.8
10. resolve that conflict by bypassing the other occurrence of $v$
11. else
12. resolve an arbitrary conflict
13. end
14. let $u$ be the newly bypassed vertex
15. end

**Output:** The Hamiltonian path $P''$.

**Rule 5.8.** *If there are only unresolved conflicts between $u$ and another bypassed vertex occurrence $t$ in $P'$, choose $v$ to be the neighbor of $u$ towards $t$.***

As with the PMCA, we will not need this rule in our examples of hard input instances.

**Lemma 5.9 [FHP06].** Procedure 5.6 returns a Hamiltonian path from $w$ to $z$ containing bypasses of total size at most 4.

**5.1.6 Completeness of the Solution**

**Lemma 5.10.** Procedure 5.6 returns a Hamiltonian path that contains all vertices in $P'$.

*Proof.* The proof is analogous to the one of Lemma 3.9.

Lemmas 5.7 and 5.10 immediately imply that $P''$ contains every vertex in $V$ at least once. On the other hand, the termination condition of Procedure 5.6 ensures that $P''$ contains every vertex in $V$ at most once.

³Note that we do not need to take care of $z$, because $z$ occurs only once in $P'$. 
5.2 An Example

To understand the algorithm better, let us consider an example. We apply the PMCA-HPP to the following family of graphs with $A = \{b_2\}$, i.e., we are looking for a shortest Hamiltonian path having $b_2$ as an endpoint.

Let $G_{3,k}(\beta)$ be the complete graph with the vertices $\{a_i, b_i, c_i \mid 1 \leq i \leq k\}$, for even $k \in \mathbb{N}$, with edge lengths $c(a_i, b_i) = c(b_i, c_i) := 1/k$, for $1 \leq i \leq k$, and $c(b_i, b_j) := 1$, for $i \neq j$, and maximum possible length for all other edges such that the $\beta$-triangle inequality is not violated.

Figure 5.1 shows the basic grid structure of the graph. (Only some edges are shown.)

![Figure 5.1: The graph $G_{3,k}(\beta)$.](image)

**Lemma 5.11.** The graph $G_{3,k}(\beta)$ satisfies the $\beta$-triangle inequality.

**Proof.** Let $\{u, v, w\}$ be a triangle, and let $e = \{u, v\}$ be the longest edge in it, i.e., the one that could be responsible for a violation. (At most one edge per triangle can violate the $\beta$-triangle inequality.) We show that there always exists another edge in this triangle that is long enough such that $e$ does not violate the $\beta$-triangle inequality.

The edge $e$ can, by construction, only be one of those edges for which we explicitly defined the edge length. Furthermore, all edges for which we did not explicitly define the length have length at least $2/k$. All edges in the entire graph thus have length at least $1/k$, and we have to prove the statement only for the edges $\{b_i, b_j\}$.

One can easily see that, independent of whether $w \in A$, $w \in B$, or $w \in C$, at least one of the two edges adjacent to $w$ has always length at least 1 and thus the triangle inequality is never violated. 

We now show one possible implementation of the PMCA-HPP. In many steps, the algorithm can choose between different possibilities. As before, we do not discuss this in every step, but just describe what our implementation chooses to do.
5.2.1 Minimum Spanning Tree

The PMCA-HPP computes the minimum spanning tree $T$ consisting of all edges for which we explicitly defined the edge cost, resulting in the set

$$U = \{a_i, c_i \mid 1 \leq i \leq k\} \cup \{b_1, b_2, b_k\},$$

consisting of all odd vertices in $T$ as well as the vertex $b_2$, because $b_2$ is in $A$ and has even degree in $T$ (see Figure 5.2).

![Figure 5.2: The minimum spanning tree $T$. The vertices in $U$ are circled.](image)

5.2.2 Minimum Path Matching

The PMCA-HPP computes the minimum path matching

\[
\{(a_1, b_1), (a_2, b_2), (c_1, b_1, b_2, c_2), (a_k, D), (b_k, c_k)\} \cup \\
\{(a_i, b_i, c_i) \mid 3 \leq i \leq k - 1\}
\]

of length $3 - 1/k$ as shown in Figure 5.3. Observe that we have introduced $2 - 1 = 1$ dummy vertex $D$.

![Figure 5.3: A minimum path matching for $U \cup \{D\}$.](image)
5.2. An Example

**Theorem 5.12.** The path matching described above is a minimum path matching for $U \cup \{D\}$ in $G_{3,k}(\beta)$.

**Proof.** Let $C_1 := \{a_1, b_1, c_1\}$, and $C_2 := \{a_2, b_2, c_2\}$. If both $b_1$ and $b_2$ are matched with vertices in $C_1$ and $C_2$, respectively, then these two paths have overall length $2/k$. The remaining path (i.e., the one connecting the yet unmatched vertices in $C_1$ and $C_2$) has length $1 + 2/k$. If both $b_1$ and $b_2$ are matched with vertices in $C_1$, then these two paths have length $1 + 2/k$ together, and the remaining path connecting $a_2$ and $c_2$ has length $2/k$. The inverse case works analogously. If $b_1$ and $b_2$ are matched with vertices in $C_1$ and $C_2$, respectively, then, from the minimality of the matching, it immediately follows that they are matched with each other with an edge of length 1. The remaining two paths connecting $a_1$ and $c_1$, and $a_2$ and $c_2$, respectively, have overall length $4/k$. So in every case, the three paths together have length $1 + 4/k$.

Furthermore observe that, among the remaining paths, at most one can have length $1/k$, because out of the two possible paths $(a_k, b_k)$ and $(b_k, c_k)$, at most one can be present in a matching.

Considering that the path matching consists of $k + 2$ paths, one of which has length 0, we obtain as a minimum length of every path matching

$$1 + 4/k + 1/k + (k - 3) \cdot 2/k = 3 - 1/k.$$  

Now we remove the path $(a_k, D)$ to obtain a path matching $\Pi$ for $U$. Observe that there are two odd vertices in $T \cup \Pi$, namely $b_2$ and $a_k$. So we do not have to remove an edge adjacent to $b_2$ in $T$.

Let $w := a_k, z := b_2$.

### 5.2.3 Conflict Resolution in the Minimum Path Matching

There are only two conflicts in $\Pi$, namely the vertices $b_1, b_2$. All paths not containing $b_1$ or $b_2$ are therefore not changed. When Procedure 5.3 encounters the conflicting paths, it calls Procedure 5.2 with parameters $\{(a_1, b_1), (a_2, b_2), (c_1, b_1, b_2, c_2)\}$ and $z = b_2$. Procedure 5.2, in turn, then sets $q := (c_1, b_1, b_2, c_2)$, and hence $v = b_1$. It then calls itself recursively with input $\{(a_1, b_1)\}$ and $v = b_1$. After this call, $q_1 = (a_1, b_1)$. Since $v = b_1$ is not internal to this path, Procedure 5.2 takes the else branch and transforms $q$ and $q_1$ into paths $q'_1 := (a_1, c_1)$ and $q' := (b_1, b_2, c_2)$.

---

4. Observe that $b_1$ and $b_2$ cannot be matched with vertices outside $C_1 \cup C_2$, as the matching then contains two paths of length at least 1 and is always longer than $\Pi$. 
The only remaining conflict is $b_2$. Again, Procedure 5.2 calls itself recursively for the path $(a_2, b_2)$, which changes nothing, i.e., $q_2 = (a_2, b_2)$ after this call. Since $b_2$ is not internal to this path, $q'$ and $q_2$ are transformed into paths $q'_2 := (a_2, b_1)$ and $q'' := (b_2, c_2)$. In the end, the paths $q'_1, q'_2$, and $q''$ form a vertex-disjoint path matching for the vertices $a_1, b_1, c_1, a_2, b_2, c_2$. The resulting vertex-disjoint path matching $\Pi'$ for the entire graph can be seen in Figure 5.4.

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{figure5-4.png}
\caption{A vertex-disjoint path matching for $U$.}
\end{figure}

5.2.4 Eulerian Path

The PMCA-HPP now computes an Eulerian path from $b_2$ to $a_k$ (see Procedure 5.4). It first checks whether there is a path in $\Pi'$ having $z = b_2$ as an endpoint. There is indeed such a path, namely $(b_2, c_2)$. It thus sets $v := c_2$. Since $\Pi'$ contains no path having $w = a_k$ as an endpoint and $T$ still contains the edge $\{w, y\}$, where $y$ is the neighbor of $w = a_k$ in $T$, i.e., $y = b_k$, it constructs an Eulerian path from $v = c_2$ to $y = b_k$ in $T - \{w, y\}$ and $\Pi' - \{(b_2, c_2)\}$, and then appends the edge $\{y, w\}$ to this path. Finally, it appends the path $(b_2, c_2)$ at the beginning of this path to obtain the Eulerian path

$$P := (b_2, c_2, b_2, a_2, b_1, c_1, b_1, b_2, b_3, c_3, a_3, b_3, b_4, \ldots, b_{k-1}, c_{k-1}, b_{k-1}, a_{k-1}, b_{k-1}, b_k, c_k, b_k, a_k)$$

as depicted in Figure 5.5. For clarity, the paths of $\Pi'$ are dashed.

5.2.5 Conflict Resolution in the Minimum Spanning Tree

The goal of the conflict resolution in the minimum spanning tree is that every vertex is adjacent to at most three edges from $T$. Instead of choosing $r$ arbitrarily as in the PMCA, $r := z = b_2$ is given. So the PMCA-HPP$_1$
5.2. An Example

bypasses, in every solid path \( p \) in Figure 5.5, the vertex \( v \) closest to \( b_2 \) if \( v \) is internal to \( p \). That is, it bypasses \( b_2 \) between \( c_2 \) and \( a_2 \) and between \( b_1 \) and \( b_3 \), and it bypasses every \( b_i \) between \( a_i \) and \( b_{i+1} \), for \( 3 \leq i \leq k - 1 \), to obtain the modified Eulerian path

\[
P := (b_2, c_2, a_2, b_1, a_1, c_1, b_3, c_3, a_3, b_4, \ldots, b_{k-1}, c_{k-1}, a_{k-1}, b_k, c_k, b_k, a_k)
\]

as shown in Figure 5.6.

5.2.6 Conflict Resolution in the Modified Eulerian Path

The goal of the last step is to bypass vertices such that the vertices \( w = a_k \) and \( z = b_2 \) have degree 1, and every other vertex has degree 2, i.e., we have a Hamiltonian path with \( b_2 \) as an endpoint. The problematic vertices are thus all \( b_i \) except \( b_2 \). Since \( w = a_k \) is no conflict, Procedure 5.6 arbitrarily chooses to bypass \( b_1 \) between \( a_2 \) and \( a_1 \). No neighbor of the bypassed occurrence is a conflict, so Procedure 5.6 chooses to bypass \( b_3 \) between \( c_3 \) and \( a_3 \), then \( b_4 \) between \( c_4 \) and \( a_4 \), and so on, until \( b_k \) is bypassed between \( c_k \) and \( a_k \). The PMCA-HPP \( 1 \) thus obtains the Hamiltonian path

\[
P'' := (b_2, c_2, a_2, a_1, c_1, b_1, b_3, c_3, a_3, b_4, \ldots, b_{k-1}, c_{k-1}, a_{k-1}, b_k, c_k, a_k)
\]
as depicted in Figure 5.7.

\[
\begin{align*}
& a_1 & a_2 & a_3 & a_{k-1} & a_k \\
& b_1 & b_2 & b_3 & b_{k-1} & b_k \\
& c_1 & c_2 & c_3 & c_{k-1} & c_k
\end{align*}
\]

**Figure 5.7:** The Hamiltonian path $P''$. 
Chapter 6

On the Approximation Ratio of the PMCA-HPP

In this chapter, we show that the upper bounds on the approximation ratio of $\frac{3}{2}\beta^2$ for the PMCA-HPP$_0$ and the PMCA-HPP$_1$ and $\frac{5}{3}\beta^2$ for the PMCA-HPP$_2$ proved by Forlizzi et al. [FHPS06] are tight, i.e., that the following holds.

**Theorem 6.1.** For every $\beta \geq 1$, there exists an infinite family of graphs satisfying the $\beta$-triangle inequality on which the PMCA-HPP$_0$ and the PMCA-HPP$_1$ cannot achieve an approximation ratio of $\frac{3}{2}\beta^2 - \varepsilon$, for any $\varepsilon > 0$.

**Theorem 6.2.** For every $\beta \geq 1$, there exists an infinite family of graphs satisfying the $\beta$-triangle inequality on which the PMCA-HPP$_2$ cannot achieve an approximation ratio of $\frac{5}{3}\beta^2 - \varepsilon$, for any $\varepsilon > 0$.

In Sections 6.1 and 6.2, we show that the graph $G_{10,k}(\beta)$ introduced in Chapter 4 contains Hamiltonian paths of a certain length with zero and one fixed endpoint, respectively. Then, we show possible implementations of the PMCA-HPP$_0$ and the PMCA-HPP$_1$ to obtain the desired lower bounds. In Section 6.3, we introduce a new graph and prove that it satisfies the $\beta$-triangle inequality. Then, we show that this graph contains a Hamiltonian path of a certain length with two fixed endpoints. Finally, we show one possible implementation of the PMCA-HPP$_2$ to obtain the desired lower bound.

### 6.1 The PMCA-HPP$_0$

Figure 6.1 shows a Hamiltonian path from $v_{3,1}$ to $v_{3,k}$ in $G_{10,k}(\beta)$. The path $(v_{3,1}, v_{6,1}, v_{2,1}, v_{1,1}, v_{4,1}, v_{8,1}, v_{9,1}, v_{10,1}, v_{5,1}, v_{7,1})$ has length

\[
1/k + \beta(1 + 1/k) + 1/k + \beta(1/k + 2\beta/k) + \beta(1/k + 2\beta/k) + 1/k + 1/k + 2\beta/k + \beta(1 + 1/k) = 2\beta + (4\beta^2 + 6\beta + 4)/k.
\]
All other paths inside a cluster, i.e., all paths of the form $(v_{1,i}, v_{2,i}, v_{4,i}, v_{8,i}, v_{9,i}, v_{10,i}, v_{5,i}, v_{6,i}, v_{3,i}, v_{7,i})$, for $2 \leq i \leq k-1$, and the path $(v_{1,k}, \ldots, v_{6,k}, v_{7,k}, v_{3,k})$, have length

$$
\frac{1}{k} + \frac{2\beta}{k} + \beta(1 + \frac{2\beta}{k}) + \frac{1}{k} + \frac{1}{k} + \frac{2\beta}{k} = 1 + (\frac{2\beta^2 + 7\beta + 4}{k}).
$$

These internal paths are connected to each other via $k-1$ edges of length $1 + \frac{2}{k}$. So the length of the depicted Hamiltonian path is

$$
2\beta + (4\beta^2 + 6\beta + 4)/k + (k - 1) \cdot (1 + (2\beta^2 + 7\beta + 4)/k + 1 + 2/k) = 2k + 2\beta^2 + 9\beta + 4 + (2\beta^2 - \beta - 2)/k.
$$

We now show one possible implementation of the PMCA-HPP\(_0\) that, on input $G_{10,k}(\beta)$, returns a Hamiltonian path of length $3(k - 2)\beta^2$.

### 6.1.1 Minimum Spanning Tree

The PMCA-HPP\(_0\) computes the same minimum spanning tree as the PMCA with all edges of length $1/k$, the edges $\{v_{5,i}, v_{6,i}\}$, for $1 \leq i \leq k$, and $\{v_{9,i}, v_{2,i+1}\}$, for $1 \leq i \leq k - 1$, as shown in Figure 6.2, resulting in the set of odd vertices

$$
U = \{v_{1,i}, v_{3,i}, v_{4,i}, v_{6,i}, v_{7,i}, v_{8,i}, v_{10,i} \mid 1 \leq i \leq k\} \cup \{v_{2,i} \mid 2 \leq i \leq k\} \cup \{v_{9,k}\}.
$$
6.1. The PMCA-HPP

The PMCA-HPP\(0\) computes the minimum path matching

\[
\{(v_1, v_2), (v_4, v_5, v_6, v_7), (v_8, v_9, v_{10}, i) \mid 2 \leq i \leq k - 1\} \cup \\
\{(v_1, v_2, v_3, v_4, v_6, v_7), (v_8, v_9, v_{10}, v_1, v_3, v_5), (v_{1, k}, v_{2, k}, v_{4, k}, v_{5, k}, v_{9, k}, v_{8, k}, v_{7, k}, v_{6, k}, v_{10, k})\}
\]

of length \(k + 4\) for \(U \cup \{D_1, D_2\}\) as described above (see Figure 6.3).

Figure 6.2: A minimum spanning tree in \(G_{10,k}(\beta)\). The odd vertices are circled.

6.1.2 Minimum Path Matching

The construction of a minimum path matching is different from this step in
the PMCA. Since there is no fixed endpoint, i.e., \(|A| = 0\), we add two dummy
vertices \(D_1\) and \(D_2\) whose adjacent edges have length 0 except for \(\{D_1, D_2\}\),
which has length \(\infty\).

The PMCA-HPP\(0\) computes the minimum path matching

\[
\{(v_1, v_2, v_3, v_4, v_5, v_6, v_7, v_8, v_9, v_{10}, v_{11}, v_{12}, v_{13}, v_{14}) \mid 2 \leq i \leq k - 1\} \cup \\
\{(v_1, v_2, v_3, v_4, v_5, v_6, v_7, v_8, v_{10}, v_1, v_3, v_5, v_6, v_7), (v_{1, k}, v_{2, k}, v_{4, k}, v_{5, k}, v_{9, k}, v_{8, k}, v_{7, k}, v_{6, k}, v_{10, k})\}
\]

of length \(k + 4\) for \(U \cup \{D_1, D_2\}\) as described above (see Figure 6.3).

Figure 6.3: A minimum path matching for \(U \cup \{D_1, D_2\}\) in \(G_{10,k}(\beta)\).
Theorem 6.3. The path matching described above is a minimum path matching for \(U \cup \{D_1, D_2\}\) in \(G_{10,k}(\beta)\), for \(k \geq 8\).

In order to prove this result, we need three lemmas.

Lemma 6.4. We can w.l.o.g. assume that a minimum path matching contains the paths \((v_{8,i}, v_{9,i}, v_{10,i})\), for \(i = 1, 2, \ldots, k\).

Proof. Let \(v_{8,i}, v_{10,i}\) be two vertices that are not matched with each other. Then, there exist w.l.o.g. paths \((v_{8,i}, v_{9,i}, \ldots, u), (v_{10,i}, v_{9,i}, \ldots, v)\) for some vertices \(u, v\). We can replace these paths with \((v_{8,i}, v_{9,i}, v_{10,i})\) and a shortest path \((u, \ldots, v_{9,i}, \ldots, v)\) that have, by construction, the same overall length. \(\square\)

Lemma 6.5. In a minimum path matching, exactly one vertex per lower subcluster is matched with a vertex that is not in this lower subcluster.

Proof. The “at least” part holds because \(|LSC_i \cap U|\) is odd for all \(i\).

Assume for contradiction that the “at most” part is wrong. Then, there is a minimum path matching that contains paths \((v_3,i, \ldots, a), (v_6,i, \ldots, b), (v_7,i, \ldots, c)\) for some vertices \(a, b, c \not\in LSC_i\). We make a case distinction according to how many vertices of \(a, b, c\) are dummy vertices and show that we can always obtain a strictly shorter path matching.

Case 1: The vertices \(a, b, c\) are no dummy vertices. This case is already covered in the proof of Lemma 4.4.

Case 2: The vertex \(a\) is a dummy vertex and the vertices \(b, c\) are not. We replace the paths \((v_3,i, D_j)\) and \((v_6,i, \ldots, b)\) of minimum length \(1 + 1/k\) with the paths \((v_3,i, v_6,i)\) and \((b, D_j)\) of length \(1/k\). The other two cases work analogously.

Case 3: The vertices \(a\) and \(b\) are dummy vertices. We replace the paths \((v_6,i, D_j)\) and \((v_7,i, \ldots, c)\) of minimum length \(1 + 2/k\) with the paths \((v_6,i, v_7,i)\) and \((c, D_j)\) of length \(1/k\). The other two cases work analogously. \(\square\)

Lemma 6.6. Let \(k \geq 8\). In a minimum path matching, only LSC vertices can be matched with the dummy vertices.

Proof. Assume for contradiction that there is a minimum path matching \(\Pi'\) that contains w.l.o.g. two edges \(\{u, D_1\}, \{v, D_2\}\) for some USC vertex \(u\) and some (LSC or USC) vertex \(v\). We know that \(c(\Pi') \leq k + 4\), because otherwise \(\Pi'\) would not be a minimum path matching. We can remove the edges \(\{u, D_1\}, \{v, D_2\}\) and add a shortest path \((u, \ldots, v)\). If \(v\) is a USC vertex, then this path has maximum length \(1 + 4/k\) (if \(u = v_{1,i}\) and \(v = v_{1,j}\) or if \(u \in \{v_{8,i}, v_{10,i}\}\) and \(v \in \{v_{8,j}, v_{10,j}\}\), for some \(i \neq j\)). If \(v\) is an
6.1. The PMCA-HPP$_0$

LSC vertex, then this path has maximum length $1 + 8/k$ (if $u \in \{v_{8,i}, v_{10,i}\}$ and $v = v_{3,j}$, for some $i \neq j$). Therefore, the path matching obtained is a path matching in $G_{10,k}(\beta)$ and has overall length at most $k + 5 + 8/k$. We know that $k \geq 8$ and thus $k + 5 + 8/k < k + 6 + 1/k$, contradicting Theorem 4.3. □

Now we are finally able to prove Theorem 6.3.

Proof. Assume for contradiction that there is a minimum path matching $\Pi'$ with length less than $k + 4$. We know from Lemma 6.4 that we can assume w.l.o.g. that $\Pi'$ contains all paths $(v_{8,i}, v_{9,i}, v_{10,i})$, for $1 \leq i \leq k$.

First consider all paths having an LSC vertex as an endpoint. We know from Lemma 6.5 that, for every LSC$_i$, this means two paths $(l_1, \ldots, l_2)$ and $(l_3, \ldots, v)$, where $l_1, l_2, l_3$ denote the three vertices in LSC$_i$ and $v$ is some vertex outside LSC$_i$. It is, however, possible that $v \in$ LSC$_j$. Let therefore $c$ be the number of paths matching two LSC vertices from different clusters.

We know from Lemma 6.6 that, for exactly two LSCs, $v$ is a dummy vertex. The length of the two paths for each of these LSCs is thus at least $1/k$. For all other LSCs, no matter what vertex we choose as $l_3$, the two paths always have a length of at least $1 + 3/k$. On the other hand, a path connecting two LSC vertices from different clusters C$_i$ and C$_j$ has a length of at least $2 + 4/k$, but we still have to consider the paths inside LSC$_i$ and LSC$_j$, respectively. Summing up, the minimum length of all paths having one or possibly two LSC vertices as endpoints amounts to

$$k \cdot 1/k + (k - 2c - 2) \cdot (1 + 2/k) + c \cdot (2 + 4/k) = k + 1 - 4/k.$$ 

Because $\Pi'$ contains all paths $(v_{8,i}, v_{9,i}, v_{10,i})$, for $1 \leq i \leq k$, the overall length of all paths having some vertex $v_{8,i}$ as an endpoint is $2$.

Now we consider all remaining paths. Because the two paths sets above contain at most $3k$ paths and the matching contains $4k + 1$ paths (the additional one because of the dummy vertices), we are left with at least $k + 1$ paths.

Let us first consider all paths of length $1/k$, i.e., all paths consisting of a single edge of length $1/k$. These paths have the form $(v_{8,i}, v_{6,i}, v_{7,i})$, for $1 \leq i \leq k$, or $(v_{1,j}, v_{2,j})$, for $2 \leq j \leq k$, or $(v_{8,k}, v_{9,k})$ or $(v_{9,k}, v_{10,k})$. But we already considered above all paths of the first two forms, and the paths of the last two forms cannot occur because of our assumption that $\Pi'$ contains all paths $(v_{8,i}, v_{9,i}, v_{10,i})$, for $1 \leq i \leq k$. So the only interesting paths are $(v_{1,j}, v_{2,j})$, for $2 \leq j \leq k$.

Let us now consider all paths of length exactly $2/k$. These paths have the form $(v_{3,i}, v_{6,i}, v_{7,i})$ or $(v_{8,i}, v_{9,i}, v_{10,i})$, for $1 \leq i \leq k$, or $(v_{2,j}, v_{5,j}, v_{4,j})$, for $2 \leq j \leq k$, or $(v_{2,k}, v_{5,k}, v_{9,k})$ or $(v_{4,k}, v_{5,k}, v_{9,k})$. But we already considered
above all paths of the first two forms, so the only interesting paths are
\((v_{2,j}, v_{5,j}, v_{4,j})\), for \(2 \leq j \leq k\), \((v_{2,k}, v_{5,k}, v_{9,k})\), and \((v_{4,k}, v_{5,k}, v_{9,k})\).

So the set of interesting paths of length at most \(2/k\) is
\[
\{(v_{1,j}, v_{2,j}), (v_{2,j}, v_{5,j}, v_{4,j}) \mid 2 \leq j \leq k\} \cup \{(v_{2,k}, v_{5,k}, v_{9,k})\} \cup \{(v_{4,k}, v_{5,k}, v_{9,k})\}.
\]

Now observe that, for some fixed \(j\) with \(2 \leq j \leq k - 1\), only one of the paths \((v_{1,j}, v_{2,j}), (v_{2,j}, v_{5,j}, v_{4,j})\) can occur in \(\Pi'\). And for \(j = k\), at most two of the paths \((v_{1,k}, v_{2,k}), (v_{2,k}, v_{5,k}, v_{4,k}), (v_{2,k}, v_{5,k}, v_{9,k}), (v_{4,k}, v_{5,k}, v_{9,k})\) can occur. That is, there are at most \(k\) remaining paths of length at most \(2/k\). In other words, there is at least one remaining path of length at least \(3/k\). Now consider the remaining \(k\) paths. From our observations above, we know that there are at most \(k - 1\) remaining paths of length \(1/k\). In other words, there is at least one remaining path of length at least \(2/k\). That is, we have at least one path of length \(3/k\) and at least one path of length \(2/k\), and \(k - 1\) remaining paths of length at least \(1/k\). Therefore, the minimum length of all these paths together is \(1 + 4/k\).

We have shown that \(\Pi'\) has minimum length
\[
k + 1 - 4/k + 2 + 1 + 4/k = k + 4,
\]
which is a contradiction to our initial assumption. \(\square\)

We remove the two paths \((v_{3,1}, D_1), (v_{3,k}, D_2)\) to obtain a minimum path matching \(\Pi\) for \(U - \{v_{3,1}, v_{3,k}\}\). There are thus two vertices of odd degree in \(\Pi \cup T\), namely \(v_{3,1}\) and \(v_{3,k}\). The PMCA-HPP\(_0\) decides to set \(w := v_{3,k}, z := v_{3,1}\).

### 6.1.3 Conflict Resolution in the Minimum Path Matching

The goal of this step is to resolve all conflicts in the minimum path matching \(\Pi\). Procedure 5.3 does this for every connected component of \(\Pi\) separately. Observe that it always chooses the else-branch because \(z\) is not contained in \(\Pi\). For the paths
\[
\{(v_{1,1}, v_{2,1}, v_{5,1}, v_{4,1}), (v_{6,1}, v_{7,1}), (v_{8,1}, v_{9,1}, v_{10,1}, v_{1,1}, v_{2,1}, v_{6,1}, v_{7,1})\} \cup \\
\{(v_{1,i}, v_{2,i}, v_{3,i}, v_{4,i}, v_{5,i}, v_{6,i}, v_{7,i}, v_{8,i}, v_{9,i}, v_{10,i}) \mid 2 \leq i \leq k - 1\},
\]
Procedure 5.2 immediately terminates because there are no conflicts.

Let us therefore look at the problematic paths of \(\Pi\), i.e.,
\[
\{(v_{1,i}, v_{5,i}, v_{6,i}, v_{3,i}, v_{4,i}), (v_{6,i}, v_{7,i}) \mid 2 \leq i \leq k - 1\} \cup \\
\{(v_{4,k}, v_{5,k}, v_{9,k}), (v_{8,k}, v_{9,k}, v_{10,k})\}.
\]
6.1. The PMCA-HPP

For the first set, Procedure 5.3 may choose \( x := v_{7,i} \). Procedure 5.2 will then choose \( v := v_{6,i} \) and will call itself recursively for the path \((v_{4,i}, v_{5,i}, v_{6,i}, v_{3,i})\) and do nothing, since a single path never has a conflict. Afterwards, it will bypass \( v \) in this path, i.e., the first set results in the vertex-disjoint paths \{\((v_{4,i}, v_{5,i}, v_{3,i}), (v_{6,i}, v_{7,i})\) \} \( 2 \leq i \leq k - 1 \).

For the second set, Procedure 5.3 may choose \( x := v_{4,k} \). Procedure 5.2 will then choose \( v := v_{9,k} \) and thus bypass \( v_{9,k} \) in the path \((v_{8,k}, v_{9,k}, v_{10,k})\), resulting in the two paths \((v_{4,k}, v_{5,k}, v_{9,k})\) and \((v_{8,k}, v_{10,k})\).

In this step, the PMCA-HPP\(_0\) thus computes the vertex-disjoint path matching

\[
\Pi' = \{(v_{1,i}, v_{2,i}), (v_{4,i}, v_{5,i}, v_{3,i}), (v_{6,i}, v_{7,i}), (v_{8,i}, v_{9,i}, v_{10,i}) \mid 2 \leq i \leq k - 1 \}\cup
\{(v_{1,i}, v_{2,i}, v_{5,i}, v_{4,i}), (v_{6,i}, v_{7,i}), (v_{8,i}, v_{9,i}, v_{10,i}),
(v_{1,k}, v_{2,k}), (v_{4,k}, v_{5,k}, v_{9,k}), (v_{6,k}, v_{7,k}), (v_{8,k}, v_{10,k})\}
\]

for \( U \) as shown in Figure 6.4.

![Figure 6.4: A vertex-disjoint path matching for \( U \) in \( G_{10,k}(\beta) \).](image)

6.1.4 Eulerian Path

Now the PMCA-HPP\(_0\) computes an Eulerian path from \( w \) to \( z \). Because there are no paths in \( \Pi' \) having \( w \) or \( z \) as an endpoint, Procedure 5.4 computes an Eulerian path from \( z \) to \( y \) in \( T - \{y, w\} \) and \( \Pi' \), where \( y \) is the neighbor of \( w \) towards \( z \) in \( T \), i.e., \( v_{6,k} \). Then, it appends the edge \( \{y, w\} \) to this path to obtain the Eulerian path

\[
P = (v_{3,1}, v_{6,1}, v_{7,1}, v_{6,1}, v_{5,1}, v_{4,1}, v_{5,1}, v_{2,1}, v_{1,1}, v_{2,1}, v_{5,1}, v_{9,1},
v_{10,1}, v_{8,1}, v_{9,1}, v_{2,2}, \ldots, v_{9,k-1}, v_{2,k}, v_{1,k}, v_{2,k}, v_{5,k},
v_{4,k}, v_{5,k}, v_{9,k}, v_{8,k}, v_{10,k}, v_{9,k}, v_{5,k}, v_{6,k}, v_{7,k}, v_{6,k}, v_{3,k})
\]
from $v_{3,1}$ to $v_{3,k}$ as shown in Figure 6.5. For clarity, the paths of $\Pi'$ are dashed.

![Figure 6.5: The Eulerian path $P$. The paths of $\Pi'$ are dashed.](image)

### 6.1.5 Conflict Resolution in the Minimum Spanning Tree

The goal of this section is to describe how the algorithm introduces some bypasses such that every vertex is adjacent to at most three edges from $T$. The PMCA-HPP$_0$ does this by considering every path $p$ in $P$ consisting only of edges from $T$ separately. If the vertex closest to $z = v_{3,1}$ in $p$ is internal to $p$, it is bypassed. Therefore, it bypasses $v_{5,1}$ between $v_{2,1}$ and $v_{9,1}, v_{5,i}$ between $v_{6,i}$ and $v_{9,i}$, for $2 \leq i \leq k - 1$, $v_{5,k}$ between $v_{9,k}$ and $v_{6,k}$, $v_{6,1}$ between $v_{7,1}$ and $v_{5,1}, v_{6,k}$ between $v_{7,k}$ and $v_{3,k}$, and $v_{9,j}$ between $v_{8,i}$ and $v_{2,j+1}$, for $1 \leq i \leq k - 1$. This results in the modified Eulerian path

$$P' = (v_{3,1}, v_{6,1}, v_{7,1}, v_{5,1}, v_{4,1}, v_{5,1}, v_{2,1}, v_{1,1}, v_{2,1}, v_{9,1}, v_{10,1}, v_{9,1}, v_{8,1}, v_{2,2}, \ldots, v_{8,k-1}, v_{2,k}, v_{1,k}, v_{2,k}, v_{5,k}, v_{4,k}, v_{5,k}, v_{9,k}, v_{8,k}, v_{10,k}, v_{9,k}, v_{6,k}, v_{7,k}, v_{3,k})$$

as shown in Figure 6.6.

### 6.1.6 Conflict Resolution in the Modified Eulerian Path

The goal of the last step is to bypass vertices such that every vertex except $w$ and $z$ has degree 2, i.e., we have a Hamiltonian path. The problematic vertices are thus the vertices $v_{2,i}, v_{5,i}, v_{9,i}$ and all $v_{6,i}$ except $v_{6,1}$ and $v_{6,k}$. Because $w = v_{3,1}$ is not a conflict in $P'$, the PMCA-HPP$_0$ starts with the resolution of an arbitrary conflict. It obtains the final Hamiltonian path

$$P'' = (v_{3,1}, v_{6,1}, v_{7,1}, v_{5,1}, v_{4,1}, v_{5,1}, v_{2,1}, v_{1,1}, v_{9,1}, v_{10,1}, v_{8,1}, v_{1,2}, \ldots, v_{8,k-1}, v_{1,k}, v_{2,k}, v_{4,k}, v_{5,k}, v_{8,k}, v_{10,k}, v_{9,k}, v_{6,k}, v_{7,k}, v_{3,k})$$
6.1. The PMCA-HPP

Figure 6.6: The modified Eulerian path $P'$. The paths of $\Pi'$ are dashed.

as shown in Figure 6.7 by doing the following:

- it bypasses $v_{2,1}$ between $v_{1,1}$ and $v_{9,1}$, and every other $v_{2,i}$ between $v_{8,i-1}$ and $v_{1,i}$;
- it bypasses $v_{5,1}$ between $v_{4,1}$ and $v_{2,1}$, $v_{5,k}$ between $v_{2,k}$ and $v_{4,k}$, and every other $v_{5,i}$ between $v_{4,i}$ and $v_{3,i}$;
- it bypasses $v_{6,i}$ between $v_{7,i}$ and $v_{9,i}$, for $2 \leq i \leq k - 1$;
- it bypasses $v_{9,k}$ between $v_{5,k}$ and $v_{8,k}$, and every other $v_{9,i}$ between $v_{10,i}$ and $v_{8,i}$.

Figure 6.7: The Hamiltonian path $P''$.

Considering only the edges $\{v_{4,i}, v_{3,i}\}, \{v_{7,i}, v_{9,i}\}, \{v_{8,i}, v_{1,i+1}\}$, for $2 \leq i \leq k - 1$, we obtain $\text{cost}(P'') \geq 3(k - 2)\beta^2$. We have thus shown that, for every $\beta \geq 1$ and arbitrarily small $\varepsilon > 0$, there exists an implementation $I$ of the PMCA-HPP such that

$$\frac{\text{cost}(I(G_{10,k}(\beta)))}{\text{Opt}_{\Delta^2\text{-HPP}}(G_{10,k}(\beta))} \geq \frac{3(k - 2)\beta^2}{2k + 2\beta^2 + 9\beta + 4 + (2\beta^2 - \beta - 2)/k} \geq \frac{3}{2}\beta^2 - \varepsilon,$$
for sufficiently large $k$, i.e., we have shown that the upper bound of $\frac{3}{2} \beta^2$ on the approximation ratio of the PMCA-HPP$_0$ is tight.

6.2 The PMCA-HPP$_1$

In this section, we consider the same graph as above, and we want to find a Hamiltonian path with one fixed endpoint $s := v_{1,1}$.

\[
\begin{align*}
&v_{1,1}, v_{4,1}, v_{8,1}, v_{1,2}, v_{4,2}, v_{8,2}, v_{1,k}, v_{4,k}, v_{8,k} \\
&v_{5,1}, v_{9,1}, v_{2,2}, v_{5,2}, v_{9,2}, \ldots, v_{2,k}, v_{5,k}, v_{9,k} \\
&v_{6,1}, v_{10,1}, v_{3,2}, v_{6,2}, v_{10,2}, \ldots, v_{3,k}, v_{6,k}, v_{10,k} \\
&v_{3,1}, v_{7,1}, v_{3,2}, v_{7,2}, v_{3,k}, v_{7,k}
\end{align*}
\]

**Figure 6.8:** A Hamiltonian path of length $2k + 2 \beta^2 + 7\beta + 5 - 2/k$ in $G_{10,k}(\beta)$ having $v_{1,1}$ as an endpoint.

Figure 6.8 shows a Hamiltonian path in $G_{10,k}(\beta)$ having $v_{1,1}$ as an endpoint. From the previous section, we know that each of the $k$ paths $(v_{1,i}, v_{2,i}, v_{4,i}, v_{8,i}, v_{9,i}, v_{10,i}, v_{5,i}, v_{6,i}, v_{3,i}, v_{7,i})$, for $1 \leq i \leq k$, has length $1 + (2 \beta^2 + 7 \beta + 4)/k$, and there are $k - 1$ edges of length $1 + 2/k$ connecting the clusters. So the length of the depicted Hamiltonian path is

\[
k \cdot (1 + (2 \beta^2 + 7 \beta + 4)/k) + (k - 1) \cdot (1 + 2/k) = 2k + 2 \beta^2 + 7 \beta + 5 - 2/k.
\]

We now show one possible implementation of the PMCA-HPP$_1$ that, on input $G_{10,k}(\beta)$ and $A := \{v_{1,1}\}$, returns a Hamiltonian path of length $3(k - 1)\beta^2$ having $v_{1,1}$ as an endpoint.

6.2.1 Minimum Spanning Tree

The PMCA-HPP$_1$ computes the same minimum spanning tree as the PMCA with all edges of length $1/k$, the edges $\{v_{5,i}, v_{6,i}\}$, for $1 \leq i \leq k$, and $\{v_{9,i}, v_{2,i+1}\}$, for $1 \leq i \leq k - 1$, as shown in Figure 6.9, resulting in the set

\[
U = \{v_{3,i}, v_{4,i}, v_{6,i}, v_{7,i}, v_{8,i}, v_{10,i} \mid 1 \leq i \leq k\} \cup \{v_{1,i}, v_{2,i} \mid 2 \leq i \leq k\} \cup \{v_{9,k}\},
\]

consisting of all odd vertices in $T$ except for $v_{1,1}$, because $v_{1,1}$ is in $A$. 


6.2. The PMCA-HPP

Assume there is a shorter path matching \( \Pi \).

Since there is one fixed endpoint, i.e., \(|A| = 1\), we add one dummy vertex \( D \) to the graph \( G_{10,k}(\beta) \) with edge costs as described in Section 5.1.1. The PMCA-HPP\(_1\) then computes the minimum path matching

\[
\{(v_{6,i}, v_{7,i}), (v_{8,i}, v_{9,i}, v_{10,i}) \mid 1 \leq i \leq k\} \cup \{(v_{4,i}, v_{5,i}, v_{6,i}, v_{3,i}) \mid 1 \leq i \leq k - 1\} \\
\cup \{(v_{1,i}, v_{2,i}) \mid 2 \leq i \leq k\} \cup \{(v_{4,k}, v_{5,k}, v_{9,k})\} \cup \{(v_{3,k}, D)\}
\]

of length \( k + 5 - 1/k \) for \( U \cup \{D\} \) in \( G_{10,k}(\beta) \) (see Figure 6.10).

\[\text{Figure 6.9: A minimum spanning tree in } G_{10,k}(\beta). \text{ The vertices in } U \text{ are circled.}\]

\[\text{Figure 6.10: A minimum path matching for } U \cup \{D\} \text{ in } G_{10,k}(\beta).\]

**Theorem 6.7.** The path matching described above is a minimum path matching for \( U \cup \{D\} \) in \( G_{10,k}(\beta) \), for every \( k \geq 8 \).

**Proof.** Assume there is a shorter path matching \( \Pi' \).
First, observe that we can assume w.l.o.g. that \((v_{8,1}, v_{9,1}, v_{10,1}) \in \Pi'\), because if this is not the case, we can just swap the endpoints of the two paths having \(v_{8,1}\) and \(v_{10,1}\) as an endpoint, respectively, to obtain a path matching containing the path \((v_{8,1}, v_{9,1}, v_{10,1})\) (cf. the proof of Lemma 6.4).

We can transform \(\Pi'\) into a path matching of length shorter than \(k + 4\) for the set \(U \cup \{D_1, D_2\}\) as defined in Section 6.1.2 as follows. Because \((v_{8,1}, v_{9,1}, v_{10,1}) \in \Pi'\), we know that there is a path \((v_{4,1}, \ldots, v)\) in \(\Pi'\) for some vertex \(v \not\in USC_1\). This path has length at least \(1 + 2/k\), and we can replace it with the two paths \((v_{4,1}, v_{5,1}, v_{2,1}, v_{1,1}), (v, D_1)\) and then just set \(D_2 := D\). So we replaced a path of length at least \(1 + 2/k\) with two paths of overall length \(3/k\), i.e., the resulting path matching for the set \(U \cup \{D_1, D_2\}\) as defined in Section 6.1.2 is strictly shorter than \(k + 5 - 1/k - (1 + 2/k) + 3/k = k + 4\), contradicting Theorem 6.3.

We remove the path \((v_{3,1}, D)\) to obtain a minimum path matching \(\Pi\) for \(U - \{v_{3,1}\}\). There are thus two vertices of odd degree in \(\Pi \cup T\), namely \(v_{1,1}\) and \(v_{3,1}\). The PMCA-HPP\(_1\) decides to set \(w := v_{3,1}, z := v_{1,1}\).

### 6.2.3 Conflict Resolution in the Minimum Path Matching

The goal of this step is to resolve all conflicts in the minimum path matching \(\Pi\). Procedure 5.3 does this for every connected component of \(\Pi\) separately. Observe that it always chooses the else-branch because \(z\) is not contained in \(\Pi\). For the paths

\[
\{(v_{8,i}, v_{9,i}, v_{10,i}) \mid 1 \leq i \leq k - 1\} \cup \{(v_{1,i}, v_{2,i}) \mid 2 \leq i \leq k\} \cup \{(v_{6,k}, v_{7,k})\},
\]

Procedure 5.2 immediately terminates because there are no conflicts.

Let us therefore now look at the problematic paths of \(\Pi\), i.e.,

\[
\{(v_{4,i}, v_{5,i}, v_{6,i}, v_{3,i}), (v_{6,i}, v_{7,i}) \mid 1 \leq i \leq k - 1\} \cup \{(v_{4,k}, v_{5,k}, v_{9,k}), (v_{8,k}, v_{9,k}, v_{10,k})\}.
\]

For the first set, Procedure 5.3 may choose \(x := v_{7,i}\). Procedure 5.2 will then choose \(v := v_{6,i}\) and will call itself recursively for the path \((v_{4,i}, v_{5,i}, v_{6,i}, v_{3,i})\) and do nothing, since a single path never has a conflict. Afterwards, it will bypass \(v\) in this path, i.e., the first set results in the vertex-disjoint paths \(\{(v_{4,i}, v_{5,i}, v_{3,i}), (v_{6,i}, v_{7,i}) \mid 1 \leq i \leq k - 1\}\).

For the second set, Procedure 5.3 may choose \(x := v_{4,k}\). Procedure 5.2 will then choose \(v := v_{9,k}\) and thus bypass \(v_{9,k}\) in the path \((v_{8,k}, v_{9,k}, v_{10,k})\), resulting in the two paths \((v_{4,k}, v_{5,k}, v_{9,k})\) and \((v_{8,k}, v_{10,k})\).
In this step, the PMCA-HPP₁ thus computes the vertex-disjoint path matching
\[
\{(v_{6,i}, v_{7,i}) \mid 1 \leq i \leq k\} \cup \{(v_{4,i}, v_{5,i}, v_{3,i}), (v_{8,i}, v_{9,i}, v_{10,i}) \mid 1 \leq i \leq k-1\} \cup \\
\{(v_{1,i}, v_{2,i}) \mid 2 \leq i \leq k\} \cup \{(v_{4,k}, v_{5,k}, v_{9,k}), (v_{8,k}, v_{10,k})\}
\]
for \(U\) as shown in Figure 6.11.

**Figure 6.11:** A vertex-disjoint path matching for \(U\) in \(G_{10,k}(\beta)\).

### 6.2.4 Eulerian Path

Now the PMCA-HPP₁ computes an Eulerian path from \(w\) to \(z\). Because there are no paths in \(\Pi'\) having \(w\) or \(z\) as an endpoint, Procedure 5.4 computes an Eulerian path from \(z\) to \(y\) in \(T - \{y, w\}\) and \(\Pi'\), where \(y\) is the neighbor of \(w\) towards \(z\) in \(T\), i.e., \(v_{6,k}\). Then, it appends the edge \(\{y, w\}\) to this path to obtain the Eulerian path
\[
P = (v_{1,1}, v_{2,1}, v_{5,1}, v_{4,1}, v_{5,1}, v_{3,1}, v_{6,1}, v_{7,1}, v_{6,1}, v_{5,1}, v_{9,1}, \\
v_{10,1}, v_{9,1}, v_{8,1}, v_{9,1}, v_{2,2}, \ldots, v_{9,k-1}, v_{2,k}, v_{1,k}, v_{2,k}, v_{5,k}, \\
v_{4,k}, v_{5,k}, v_{9,k}, v_{8,k}, v_{10,k}, v_{9,k}, v_{5,k}, v_{6,k}, v_{7,k}, v_{6,k}, v_{3,k})
\]
from \(v_{1,1}\) to \(v_{3,k}\) as shown in Figure 6.12. For clarity, the paths of \(\Pi'\) are dashed.

### 6.2.5 Conflict Resolution in the Minimum Spanning Tree

The goal of this section is to describe how the algorithm introduces some bypasses such that every vertex is adjacent to at most three edges from \(T\). The PMCA-HPP₁ does this by considering every path \(p\) in \(P\) consisting
The goal of the last step is to bypass vertices such that every vertex except endpoint. The problematic vertices are thus

$$\{v_{2,i} \mid 2 \leq i \leq k\} \cup \{v_{5,i}, v_{9,i} \mid 1 \leq i \leq k\} \cup \{v_{6,i} \mid 1 \leq i \leq k - 1\}.$$
Because \( w = v_{3,k} \) is not a conflict in \( P' \), the PMCA-HPP\(_1\) starts with the resolution of an arbitrary conflict. It obtains the final Hamiltonian path

\[
P'' = (v_{1,1}, v_{2,1}, v_{5,1}, v_{4,1}, v_{3,1}, v_{6,1}, v_{7,1}, v_{9,1}, v_{10,1}, v_{8,1}, v_{1,2}, \ldots, v_{1,k}, v_{2,k}, v_{5,k}, v_{4,k}, v_{9,k}, v_{8,k}, v_{10,k}, v_{6,k}, v_{7,k}, v_{3,k})
\]

as shown in Figure 6.14 by doing the following:

- it bypasses \( v_{2,j} \) between \( v_{8,i-1} \) and \( v_{1,i} \), for \( 2 \leq i \leq k \);
- it bypasses \( v_{5,k} \) between \( v_{4,k} \) and \( v_{9,k} \), and every other \( v_{5,i} \) between \( v_{4,i} \) and \( v_{3,i} \);
- it bypasses \( v_{6,j} \) between \( v_{7,i} \) and \( v_{9,i} \), for \( 1 \leq i \leq k - 1 \);
- it bypasses \( v_{9,k} \) between \( v_{10,k} \) and \( v_{6,k} \), and every other \( v_{9,i} \) between \( v_{10,i} \) and \( v_{8,i} \).

![Figure 6.14: The Hamiltonian path \( P'' \).](image)

Considering only the edges \( \{v_{4,i}, v_{3,i}\}, \{v_{7,i}, v_{9,i}\}, \{v_{8,i}, v_{1,i+1}\} \), for \( 1 \leq i \leq k - 1 \), we obtain \( \text{cost}(P'') \geq 3(k - 1)\beta^2 \). We have thus shown that, for every \( \beta \geq 1 \) and arbitrarily small \( \epsilon > 0 \), there exists an implementation \( I \) of the PMCA-HPP\(_1\) such that

\[
\frac{\text{cost}(I(G_{10,k}(\beta)))}{\text{Opt}_{\Delta_{\text{HPP}}}(G_{10,k}(\beta))} \geq \frac{3(k - 1)\beta^2}{2k + 2\beta^2 + 7\beta + 5 - 2/k} \geq \frac{3}{2}\beta^2 - \epsilon,
\]

for sufficiently large \( k \), i.e., we have shown that the upper bound of \( \frac{3}{2}\beta^2 \) on the approximation ratio of the PMCA-HPP\(_1\) is tight.
6.3 The PMCA-HPP\textsubscript{2}

In the previous two sections, we could reuse the worst-case example from Chapter 4. For the PMCA-HPP\textsubscript{2}, however, this is not possible, as we shall see.

Observe that, for $\beta = 1$, there exists an implementation of the PMCA that is also an implementation of the Christofides algorithm. This implementation does not construct a path matching in the second step, but a normal matching.\textsuperscript{1}

Observe furthermore that, for $\beta = 1$, there exists an implementation of the PMCA-HPP\textsubscript{2} that is also an implementation of Hoogeveen’s algorithm. As above, it is necessary always to construct a matching instead of a path matching.

Assume now that we could use the graph $G_{10,k}(\beta)$ to establish a lower bound of $\frac{5}{3} - \varepsilon$ on the approximation ratio of the PMCA-HPP\textsubscript{2}. In particular, we could show an implementation that achieves an approximation ratio of $\frac{5}{3}$ on the graph $G_{10,k}(1)$. This, in turn, contradicts the fact that the sets of worst-case instances for the metric TSP and the metric HPP\textsubscript{2} are disjoint [Möm11].

So in order to prove the lower bound of $\frac{5}{3} - \varepsilon$, we need to introduce a new graph. Let $G_{18,k}(\beta)$ be the complete graph with the vertices $\{v_{i,j} \mid 1 \leq i \leq 18, 1 \leq j \leq k\}$, for $k \in \mathbb{N}$, with edge lengths

\[
c(v_{1,i}, v_{4,i}) = c(v_{2,i}, v_{4,i}) = c(v_{3,i}, v_{4,i}) = c(v_{5,i}, v_{8,i}) = c(v_{6,i}, v_{8,i}) = c(v_{7,i}, v_{8,i}) = c(v_{9,i}, v_{12,i}) = c(v_{10,i}, v_{12,i}) = c(v_{11,i}, v_{12,i}) = c(v_{13,i}, v_{14,i}) = c(v_{14,i}, v_{18,i}) = c(v_{15,i}, v_{18,i}) = c(v_{16,i}, v_{18,i}) = c(v_{17,i}, v_{18,i}) = c(v_{18,i}, v_{12,j+1}) = 1/k, \\
c(v_{4,i}, v_{12,i}) = c(v_{8,i}, v_{18,i}) := 1, \\
c(v_{3,i}, v_{5,i}) = c(v_{12,i}, v_{14,i}) := 1 + 1/k,
\]

for $1 \leq i \leq k, 1 \leq j \leq k - 1$, and maximum possible length for all other edges such that the $\beta$-triangle inequality is not violated. Figure 6.15 shows the basic structure of the graph. (Only some edges are shown.) Observe that the graph consists of $k$ clusters, each consisting of eighteen vertices. We denote these clusters by $C_i$, i.e., $C_i := \{v_{1,i}, v_{2,i}, \ldots, v_{18,i}\}$, for $1 \leq i \leq k$. Furthermore, we call the set $\{v_{1,i}, v_{2,i}, v_{3,i}, v_{4,i}\}$ the upper left subcluster $i$, denoted $ULSC_i$, the vertex set $\{v_{5,i}, v_{6,i}, v_{7,i}, v_{8,i}\}$ the upper right subcluster $i$, denoted $URSC_i$,

\textsuperscript{1}The condition $\beta = 1$ is necessary to ensure that no path matching is strictly shorter than a minimum matching.
6.3. The PMCA-HPP

and the vertex set

\[ \{v_{13,j}, v_{14,j}, v_{15,j}, v_{16,j}, v_{17,j}, v_{18,j}, v_{9,j+1}, v_{10,j+1}, v_{11,j+1}, v_{12,j+1}\} \]

the lower subcluster \( j \), denoted \( LSC_j \), for \( 1 \leq i \leq k, 1 \leq j \leq k - 1 \). Furthermore, let \( v_{13,k}, v_{14,k}, v_{15,k}, v_{16,k}, v_{17,k}, v_{18,k} \) be the lower subcluster \( k \), denoted \( LSC_k \).\(^2\) The vertices are called ULSC vertices, URSC vertices, and LSC vertices, respectively.

[Figure 6.15: The graph \( G_{18,k}(\beta) \).

**Lemma 6.8.** The graph \( G_{18,k}(\beta) \) satisfies the \( \beta \)-triangle inequality.

**Proof.** Let \( \{u, v, w\} \) be a triangle, and let \( e = \{u, v\} \) be the longest edge in it, i.e., the one that could be responsible for a violation. We show that there always exists another edge in this triangle that is long enough such that \( e \) does not violate the \( \beta \)-triangle inequality.

The edge \( e \) can, by construction, only be one of those edges for which we explicitly define the edge length. Furthermore, all edges for which we did not explicitly define the length have length at least \( 2/k \). All edges in the entire graph thus have length at least \( 1/k \), and we have to prove the statement only in the following four cases.

**Case 1:** \( e = \{v_{3,i}, v_{5,i}\} \). If \( w \in ULSC_i \), then the shortest path from \( w \) to \( v_{3,i} \) has length at least 1, and thus also the edge \( \{w, v_{3,i}\} \). Therefore, \( c(v_{3,i}, v_{5,i}) = 1 + 1/k \leq \beta(1/k + 1) \leq \beta(c(v_{3,i}, w) + c(w, v_{5,i})) \). Otherwise, the shortest path from \( v_{3,i} \) to \( w \) has length at least 1, and thus also the edge \( \{v_{3,i}, w\} \). Therefore, \( c(v_{3,i}, v_{5,i}) = 1 + 1/k \leq \beta(1 + 1/k) \leq \beta(c(v_{3,i}, w) + c(w, v_{5,i})) \).

\(^2\)Note that, for \( 1 \leq j \leq k - 1 \), the vertex set \( LSC_j \) contains vertices from both \( C_j \) and \( C_{j+1} \), so in particular \( LSC_j \not\subseteq C_j \).
Case 2: \( e = \{v_{4,i}, v_{12,i}\} \). If \( w \in ULSC_i \), then the shortest path from \( w \) to \( v_{12,i} \) has length at least 1, and thus also the edge \( \{w, v_{12,i}\} \). Therefore, 
\[ c(v_{4,i}, v_{12,i}) = 1 \leq \beta(1/k + 1) \leq \beta(c(v_{4,i}, w) + c(w, v_{12,i})) \]. Otherwise, the shortest path from \( v_{4,i} \) to \( w \) has length at least 1, and thus also the edge \( \{v_{4,i}, w\} \). Therefore, 
\[ c(v_{4,i}, v_{12,i}) = 1 \leq \beta(1 + 1/k) \leq \beta(c(v_{4,i}, w) + c(w, v_{12,i})) \].

Case 3: \( e = \{v_{8,i}, v_{18,i}\} \). If \( w \in URSC_i \), then the shortest path from \( w \) to \( v_{18,i} \) has length at least 1, and thus also the edge \( \{w, v_{18,i}\} \). Therefore, 
\[ c(v_{8,i}, v_{18,i}) = 1 \leq \beta(1/k + 1) \leq \beta(c(v_{8,i}, w) + c(w, v_{18,i})) \]. Otherwise, the shortest path from \( v_{8,i} \) to \( w \) has length at least 1, and thus also the edge \( \{v_{8,i}, w\} \). Therefore, 
\[ c(v_{8,i}, v_{18,i}) = 1 \leq \beta(1 + 1/k) \leq \beta(c(v_{8,i}, w) + c(w, v_{18,i})) \].

Case 4: \( e = \{v_{12,i}, v_{14,i}\} \). If \( w \in LSC_i \), then the shortest path from \( v_{12,i} \) to \( w \) has length at least 1, and thus also the edge \( \{v_{12,i}, w\} \). Therefore, 
\[ c(v_{12,i}, v_{14,i}) = 1 + 1/k \leq \beta(1 + 1/k) \leq \beta(c(v_{12,i}, w) + c(w, v_{14,i})) \]. Otherwise, the shortest path from \( w \) to \( v_{14,i} \) has length at least 1, and thus also the edge \( \{w, v_{14,i}\} \). Therefore, 
\[ c(v_{12,i}, v_{14,i}) = 1 + 1/k \leq \beta(1/k + 1) \leq \beta(c(v_{12,i}, w) + c(w, v_{14,i})) \].

\[ \square \]

**Figure 6.16:** A Hamiltonian path in \( G_{18,k}(\beta) \) from \( v_{9,1} \) to \( v_{17,k} \) of length \( 3k + 2\beta^2 + 21\beta + 5 - \frac{2\beta^2 + 3\beta}{k} \).

Figure 6.16 shows a Hamiltonian path from \( v_{9,1} \) to \( v_{17,k} \) in \( G_{18,k}(\beta) \). This path consists of \( k \) sequences

\[ v_{9,i}, v_{10,i}, v_{11,i}, v_{12,i}, v_{13,i}, v_{14,i}, v_{15,i}, v_{16,i}, v_{17,i}, \]

\[ v_{6,i}, v_{7,i}, v_{8,i}, v_{18,i}, v_{19,i}, v_{20,i}, v_{21,i}, v_{22,i}, v_{23,i}, \]

for \( 1 \leq i \leq k \), internal to each cluster. Each such sequence has length

\[ \frac{2\beta}{k} + \frac{2\beta^2}{k} + 1/k + 1 + 1/k + \frac{2\beta}{k} + \frac{2\beta}{k} + 1 + 1/k + \frac{2\beta}{k} + \frac{2\beta}{k} = 3 + \frac{20\beta + 5}{k}. \]
Additionally, the path contains \( k - 1 \) sequences \( v_{17,i}, v_{9,i+1} \), for \( 1 \leq i \leq k - 1 \), of length \( (2\beta^2 + \beta)/k \) each.

So the length of the depicted Hamiltonian path is

\[
k \cdot \left( 3 + \frac{20\beta + 5}{k} \right) + (k - 1) \cdot \left( \frac{2\beta^2 + \beta}{k} \right) = 3k + 2\beta^2 + 21\beta + 5 - \frac{2\beta^2 + \beta}{k}.
\]

We now show one possible implementation of the PMCA-HPP\(_2\). Additionally, the path contains sequences \( v_{17,i}, v_{9,i+1} \), for \( 1 \leq i \leq k - 1 \), of length \( (2\beta^2 + \beta)/k \) each.

6.3.1 Minimum Spanning Tree

The PMCA-HPP\(_2\) computes the minimum spanning tree with all edges of length 1, all edges of length \( 1/k \) adjacent to the vertices \( v_{4,i}, v_{8,i}, v_{12,i}, v_{13,i}, v_{18,i} \), for \( 1 \leq i \leq k \), as well as the edges \( v_{12,i}, v_{14,i} \), for \( 1 \leq i \leq k \), as shown in Figure 6.17, resulting in the set

\[
U = \{v_{1,i}, v_{2,i}, v_{3,i}, v_{5,i}, v_{6,i}, v_{7,i}, v_{10,i}, v_{11,i}, v_{13,i}, v_{14,i}, v_{15,i}, v_{16,i} | 1 \leq i \leq k \} \cup \{v_{9,i} | 2 \leq i \leq k \} \cup \{v_{12,1}, v_{18,k}\} \cup \{v_{17,i} | 1 \leq i \leq k - 1 \},
\]

consisting of all odd vertices in \( T \) except for the two vertices \( v_{9,1} \) and \( v_{17,k} \) that are in \( A \).

![Figure 6.17](image)

**Figure 6.17**: A minimum spanning tree in \( G_{18,k}(\beta) \). The vertices in \( U \) are circled.

It is easy to see that \( T \) is indeed a minimum spanning tree. Every edge in the graph has length at least \( 1/k \), therefore the edges of length \( 1/k \) adjacent to \( v_{4,i}, v_{8,i} \), for \( 1 \leq i \leq k \), \( v_{13,i}, v_{18,i}, v_{12,i+1} \), for \( 1 \leq i \leq k - 1 \), and \( v_{12,1} \), form minimum spanning trees for the upper left, upper right, and lower subcluster \( i \), respectively. All we have to do is add an edge for every component such

\footnote{The same holds of course for the edges \( \{v_{9,1}, v_{12,1}\}, \{v_{10,1}, v_{12,1}\}, \{v_{11,1}, v_{12,1}\} \) and the vertex set \( \{v_{9,1}, v_{10,1}, v_{11,1}, v_{12,1}\} \).}
that it is connected to the rest in the end. All edges available for this have length at least 1, so we can just take the edges \{v_{4,i}, v_{12,i}\} and \{v_{8,i}, v_{18,i}\}, for \(1 \leq i \leq k\). After that, all edges available to connect the different components have length at least \(1 + 1/k\), so we can just take the edges \{v_{12,i}, v_{14,i}\}, for \(1 \leq i \leq k\).

### 6.3.2 Minimum Path Matching

Since there are two fixed endpoints, i.e., \(|A| = 2\), we add no dummy vertex and the graph remains as it is. The PMCA-HPP\(_2\) computes the minimum path matching

\[
\Pi = \{(v_{1,1}, v_{4,1}, v_{12,1}), (v_{2,1}, v_{4,1}, v_{3,1}), (v_{10,1}, v_{12,1}, v_{11,1})\} \cup \\
\{(v_{5,i}, v_{8,i}, v_{6,i}), (v_{7,i}, v_{8,i}, v_{18,i}, v_{17,i}), (v_{13,i}, v_{14,i}), (v_{15,i}, v_{18,i}, v_{16,i}) \mid 1 \leq i \leq k - 1\} \cup \\
\{(v_{1,i}, v_{4,i}, v_{12,i}, v_{9,i}), (v_{2,i}, v_{4,i}, v_{3,i}), (v_{10,i}, v_{12,i}, v_{11,i}) \mid 2 \leq i \leq k\} \cup \\
\{(v_{5,k}, v_{8,k}, v_{6,k}), (v_{7,k}, v_{8,k}, v_{18,k}, v_{15,k}), (v_{13,k}, v_{14,k}), (v_{16,k}, v_{18,k})\}
\]

of length \(2k + 13 - 2/k\) for \(U\) in \(G_{18,k}(\beta)\) (see Figure 6.18).

![Figure 6.18: A minimum path matching for U in G_{18,k}(\beta).](image)

**Theorem 6.9.** \(\Pi\) is a minimum path matching for \(U\) in \(G_{18,k}(\beta)\).

The proof is by contradiction. Let us assume there is a minimum path matching \(\Pi'\) for \(U\) that has cost less than \(2k + 13 - 2/k\).

**Lemma 6.10.** W.l.o.g., \(\Pi'\) contains the paths \((v_{13,i}, v_{14,i})\), for \(1 \leq i \leq k\).

**Proof.** Assume that there are two vertices \(v_{13,i}, v_{14,i}\) that are not matched with each other in \(\Pi'\). We can w.l.o.g. assume that the shortest path having \(v_{13,i}\) as an endpoint has the form \((v_{13,i}, v_{14,i}, \ldots, u)\). Let \(v\) be the vertex matched with \(v_{14,i}\). Replace the two paths with the paths \((v_{13,i}, v_{14,i})\) and \((u, \ldots, v_{14,i}, \ldots, v)\) of the same overall length. \(\square\)
6.3. The PMCA-HPP

Lemma 6.11. W.l.o.g., $v_{10,i}$ is matched with $v_{11,i}$ and $v_{15,i}$ with $v_{16,i}$ in $\Pi'$, for $1 \leq i \leq k$.

Proof. We first show that two of the three vertices $v_{9,i}, v_{10,i}, v_{11,i}$ are matched with each other, for $2 \leq i \leq k$.

Assume for contradiction there are three different paths having one of the three vertices as an endpoint each. We can w.l.o.g. assume that these paths have the form $(v_{9,i}, v_{12,i}, \ldots, u), (v_{10,i}, v_{12,i}, \ldots, v), (v_{11,i}, v_{12,i}, \ldots, w)$. We can replace the first two paths with the paths $(v_{9,i}, v_{12,i}, v_{10,i})$ and $(u, \ldots, v_{12,i}, \ldots, v)$ of the same overall length. A similar argument shows that two of the three vertices $v_{10,1}, v_{11,1}, v_{12,1}$ are matched with each other.

Assume now for contradiction that $v_{10,1}$ or $v_{11,1}$ is matched with $v_{12,1}$. As in the proof of Lemma 4.6, we can flip the endpoints of the two paths having $v_{10,1}$ and $v_{11,1}$, respectively, as endpoints to obtain a path matching that contains the path $(v_{10,1}, v_{12,1}, v_{11,1})$.

For $i > 1$, assume for contradiction that $v_{9,i}$ is matched with $v_{10,i}$ or $v_{11,i}$ in $\Pi'$, i.e., assume that w.l.o.g. $\Pi'$ contains the path $(v_{9,i}, v_{12,i}, v_{10,i})$ or $(v_{9,i}, v_{12,i}, v_{11,i})$ and the path $(v_{11,i}, v_{12,i}, \ldots, v)$ or $(v_{10,i}, v_{12,i}, \ldots, v)$, respectively. We can again flip the endpoints to obtain the paths $(v_{10,i}, v_{12,i}, v_{11,i})$ and $(v_{9,i}, v_{12,i}, \ldots, v)$ of the same overall length.

The proof is analogous for $v_{15,i}$ and $v_{16,i}$.

Lemma 6.12. W.l.o.g., $v_{1,i}$ is matched with $v_{2,i}$ and $v_{6,i}$ with $v_{7,i}$ in $\Pi'$, for $1 \leq i \leq k$.

Proof. The proof is very similar to the previous one. As above, we prove only the statement for the vertices $v_{1,i}$ and $v_{2,i}$. The proof is analogous for $v_{6,i}$ and $v_{7,i}$.

Again, we first show that two of the three vertices $v_{1,i}, v_{2,i}, v_{3,i}$ are matched with each other.

Assume for contradiction there are three different paths having one of the three vertices as an endpoint each. We can w.l.o.g. assume that these paths have the form $(v_{1,i}, v_{4,i}, \ldots, u), (v_{2,i}, v_{4,i}, \ldots, v), (v_{3,i}, \ldots, w)$. Observe that we cannot w.l.o.g. assume that the third path contains the vertex $v_{4,i}$, as e.g. the shortest path from $v_{3,i}$ to $v_{5,i}$ only consists of the edge $\{v_{3,i}, v_{5,i}\}$. We can replace the first two paths with the paths $(v_{1,i}, v_{4,i}, v_{2,i})$ and $(u, \ldots, v_{4,i}, \ldots, v)$ of the same overall length.

Assume now for contradiction that $v_{1,i}$ or $v_{2,i}$ is matched with $v_{3,i}$ in $\Pi'$, i.e., we can assume w.l.o.g. that $\Pi'$ contains the path $(v_{1,i}, v_{4,i}, v_{3,i})$ or $(v_{2,i}, v_{4,i}, v_{3,i})$ and the path $(v_{2,i}, v_{4,i}, \ldots, v)$ or $(v_{1,i}, v_{4,i}, \ldots, v)$, respectively. We can replace these two paths with the paths $(v_{1,i}, v_{4,i}, v_{2,i}), (v_{3,i}, v_{4,i}, \ldots, v)$ of the same overall length.
Observe that the three proofs above do not interfere with each other, i.e.,
we can indeed obtain a minimum path matching that satisfies the conditions of
all three lemmas by applying the transformations described in the respective
proofs. In other words, we can assume w.l.o.g. that \( \Pi' \) contains all the paths
\((v_{1,i}, v_{4,i}, v_{2,i}), (v_{6,i}, v_{8,i}, v_{7,i}), (v_{10,i}, v_{12,i}, v_{11,i}), (v_{13,i}, v_{14,i}), (v_{15,i}, v_{18,i}, v_{16,i})\).

So the remaining unmatched vertices in \( U \) are
\[
\{v_{3,i}, v_{5,i} \mid 1 \leq i \leq k\} \cup \{v_{12,1}, v_{18,k}\} \cup \{v_{9,i} \mid 2 \leq i \leq k\} \cup \{v_{17,i} \mid 1 \leq i \leq k - 1\}.
\]

Note that this means exactly four vertices per cluster remain unmatched.

**Lemma 6.13.** \( \Pi' \) contains no path with endpoints in different clusters.

**Proof.** We use induction. Assume for contradiction that there are two vertices
in \( C_1 \) that are matched with vertices outside \( C_1 \). We can w.l.o.g. assume that
they both use the edge \( \{v_{18,1}, v_{12,2}\} \). Therefore, \( \Pi' \) is not edge-disjoint and
thus not minimal. The same argument can be applied inductively to every
cluster, using the induction hypothesis that tells us that cluster \( i \) is not
connected to any cluster \( j \), for \( j < i \). \( \square \)

So we only have to find a minimum path matching for each cluster, i.e.,
for the vertices \( v_{3,1}, v_{5,1}, v_{12,1}, v_{17,1} \) in \( C_1 \), for the vertices \( v_{3,i}, v_{5,i}, v_{9,i}, v_{17,i} \)
in the clusters \( 2 \) to \( k - 1 \), and for the vertices \( v_{3,k}, v_{5,k}, v_{9,k}, v_{18,k} \) in \( C_k \). It can
easily be seen that the path matching
\[
\{(v_{3,1}, v_{4,1}, v_{12,1}), (v_{5,1}, v_{8,1}, v_{18,1}, v_{17,1})\} \cup \{(v_{3,i}, v_{4,i}, v_{12,i}, v_{9,i}) \mid 2 \leq i \}\cup
\{(v_{5,i}, v_{8,i}, v_{18,i}, v_{17,i} \mid 2 \leq i \leq k - 1\} \cup \{(v_{5,k}, v_{8,k}, v_{18,k})\}
\]
is such a minimum path matching, and we can therefore w.l.o.g. assume that
\( \Pi' \) contains exactly these paths. They have overall length
\[
1 + 1/k + 1 + 2/k + (k - 1) \cdot (1 + 2/k) + (k - 2) \cdot (1 + 2/k) + 1 + 1/k = 2k + 4 - 2/k.
\]

Let us now compute the length of \( \Pi' \). There are \( k \) paths \((v_{13,i}, v_{14,i})\) of length
\( 1/k, 2k \) paths \((v_{10,i}, v_{12,i}, v_{11,1})\) and \((v_{15,i}, v_{18,i}, v_{16,i})\) of length \( 2/k \) each, another
\( 2k \) paths \((v_{1,i}, v_{4,i}, v_{2,i})\) and \((v_{6,i}, v_{8,i}, v_{7,i})\) of length \( 2/k \) each, and the path
matching described in the previous paragraph of length \( 2k + 4 - 2/k \). In total, we get
\[
k \cdot 1/k + 2k \cdot 2/k + 2k \cdot 2/k + 2k + 4 - 2/k = 2k + 13 - 2/k,
\]
which is a contradiction to our initial assumption. This concludes the proof
of Theorem 6.9.

There are two odd vertices in the graph \( T \cup \Pi \), namely \( v_{9,1} \) and \( v_{17,k} \). The
PMCA-HPP \(_2\) decides to set \( w := v_{17,k}, z := v_{9,1} \).
6.3. The PMCA-HPP2

6.3.3 Conflict Resolution in the Minimum Path Matching

The goal of this step is to resolve all conflicts in the minimum path matching Π. Procedure 5.3 does this for every connected component of Π separately. Observe that it always chooses the else-branch because z is not contained in Π. For the paths \{(v_{13,i}, v_{14,i}) \mid 1 \leq i \leq k\}, Procedure 5.2 immediately terminates because there are no conflicts. Let us therefore now look at the problematic paths of Π, i.e.,

\[
\{(v_{1,1}, v_{4,1}, v_{12,1}), (v_{2,1}, v_{4,1}, v_{3,1}), (v_{10,1}, v_{12,1}, v_{11,1})\} \cup
\{(v_{5,i}, v_{8,i}, v_{6,i}), (v_{7,i}, v_{8,i}, v_{18,i}, v_{17,i}), (v_{15,i}, v_{18,i}, v_{16,i}) \mid 1 \leq i \leq k - 1\} \cup
\{(v_{1,1}, v_{4,i}, v_{12,i}, v_{9,i}), (v_{2,i}, v_{4,i}, v_{3,i}), (v_{10,i}, v_{12,i}, v_{11,i}) \mid 2 \leq i \leq k\} \cup
\{(v_{5,k}, v_{8,k}, v_{6,k}), (v_{7,k}, v_{8,k}, v_{18,k}, v_{15,k}), (v_{16,k}, v_{18,k})\}.
\]

For the first set, Procedure 5.3 may choose \(x := v_{2,1}\). Procedure 5.2 will then choose \(v := v_{4,1}\) and will call itself recursively for the paths \((v_{1,1}, v_{4,1}, v_{12,1}), (v_{10,1}, v_{12,1}, v_{11,1})\). This results in the vertex \(v_{12,1}\) being bypassed in the latter path. Then, Procedure 5.2 will bypass \(v_{4,1}\) in the path \((v_{1,1}, v_{4,1}, v_{12,1})\), i.e., the first set results in the vertex-disjoint paths \((v_{1,1}, v_{12,1}), (v_{2,1}, v_{4,1}, v_{3,1}), (v_{10,1}, v_{11,1})\).

For the second set, Procedure 5.3 may choose \(x := v_{6,i}\). Procedure 5.2 will then choose \(v := v_{8,i}\) and will call itself recursively for the paths \((v_{7,i}, v_{8,i}, v_{18,i}, v_{17,i}), (v_{15,i}, v_{18,i}, v_{16,i})\). This results in the vertex \(v_{18,i}\) being bypassed in the latter path. Then, Procedure 5.2 will bypass \(v_{8,i}\) in the path \((v_{7,i}, v_{8,i}, v_{18,i}, v_{17,i})\), i.e., the second set results in the vertex-disjoint paths \((v_{7,i}, v_{8,i}, v_{18,i}, v_{17,i}), (v_{15,i}, v_{16,i})\), for \(1 \leq i \leq k - 1\).

For the third set, Procedure 5.3 may choose \(x := v_{2,i}\). Procedure 5.2 will then choose \(v := v_{4,i}\) and will call itself recursively for the paths \((v_{1,i}, v_{4,i}, v_{12,i}, v_{9,i}), (v_{10,i}, v_{12,i}, v_{11,i})\). This results in the vertex \(v_{12,i}\) being bypassed in the latter path. Then, Procedure 5.2 will bypass \(v_{4,i}\) in the path \((v_{1,i}, v_{4,i}, v_{12,i}, v_{9,i})\), i.e., the third set results in the vertex-disjoint paths \((v_{1,i}, v_{12,i}, v_{9,i}), (v_{2,i}, v_{4,i}, v_{3,i}), (v_{10,i}, v_{11,i})\), for \(2 \leq i \leq k\).

For the fourth set, Procedure 5.3 may choose \(x := v_{6,k}\). Procedure 5.2 will then choose \(v := v_{8,k}\) and will call itself recursively for the paths \((v_{7,k}, v_{8,k}, v_{18,k}, v_{15,k}), (v_{16,k}, v_{18,k})\). It will once again call itself recursively for the path \((v_{16,k}, v_{18,k})\), which changes nothing because a single path has no conflict. The next step, however, is different to the three sets above. Because \(v_{18,k}\) is not internal to \((v_{6,k}, v_{18,k})\), the two paths \((v_{7,k}, v_{8,k}, v_{18,k}, v_{15,k}), (v_{16,k}, v_{18,k})\) are transformed into the paths \((v_{15,k}, v_{16,k}), (v_{7,k}, v_{8,k}, v_{18,k})\). Finally, \(v_{8,k}\) is bypassed in the latter path, i.e., the fourth set results in the vertex-disjoint
paths \((v_{5,k}, v_{8,k}, v_{6,k}), (v_{7,k}, v_{18,k}), (v_{15,k}, v_{16,k})\).

In this step, the PMCA-HPP\(_2\) thus computes the vertex-disjoint path matching

\[
\Pi = \{(v_{1,1}, v_{12,1}), (v_{2,1}, v_{4,1}, v_{3,1}), (v_{10,1}, v_{11,1})\} \cup \\
\{(v_{5,i}, v_{8,i}, v_{6,i}), (v_{7,i}, v_{18,i}, v_{17,i}), (v_{13,i}, v_{14,i}), (v_{15,i}, v_{16,i}) \mid 1 \leq i \leq k - 1\} \cup \\
\{(v_{1,i}, v_{12,i}, v_{9,i}), (v_{2,i}, v_{4,i}, v_{3,i}), (v_{10,i}, v_{11,i}) \mid 2 \leq i \leq k\} \cup \\
\{(v_{5,k}, v_{8,k}, v_{6,k}), (v_{7,k}, v_{18,k}), (v_{13,k}, v_{14,k}), (v_{15,k}, v_{16,k})\}
\]

for \(U\) as shown in Figure 6.19.

![Figure 6.19: A vertex-disjoint path matching for \(U\) in \(G_{18,k}(\beta)\).](image)

### 6.3.4 Eulerian Path

Now the PMCA-HPP\(_2\) computes an Eulerian path from \(w\) to \(z\). Because there are no paths in \(\Pi'\) having \(w\) or \(z\) as an endpoint, Procedure 5.4 computes an Eulerian path from \(z\) to \(y\) in \(T - \{y, w\}\) and \(\Pi'\), where \(y\) is the neighbor of \(w\) towards \(z\) in \(T\), i.e., \(v_{18,k}\). Then, it appends the edge \(\{y, w\}\) to this path to obtain the Eulerian path

\[
P = (v_{9,1}, v_{12,1}, v_{4,1}, v_{2,1}, v_{4,1}, v_{3,1}, v_{1,1}, v_{12,1}, v_{10,1}, v_{11,1}, v_{12,1}, v_{14,1}, \\
v_{13,1}, v_{14,1}, v_{18,1}, v_{15,1}, v_{16,1}, v_{18,1}, v_{8,1}, v_{6,1}, v_{8,1}, v_{5,1}, v_{8,1}, v_{7,1}, v_{18,1}, \\
v_{17,1}, v_{18,1}, v_{12,2}, v_{11,2}, v_{10,2}, v_{12,2}, v_{4,2}, v_{2,2}, v_{4,2}, v_{3,2}, v_{12,2}, v_{12,2}, \\
v_{9,2}, v_{12,2}, v_{14,2}, v_{13,2}, v_{14,2}, v_{18,2}, v_{15,2}, v_{16,2}, v_{18,2}, v_{8,2}, v_{6,2}, v_{8,2}, v_{5,2}, \\
v_{8,2}, v_{7,2}, v_{18,2}, v_{17,2}, v_{18,2}, v_{12,3}, \ldots, v_{12,k}, v_{11,k}, v_{10,k}, v_{12,k}, v_{4,k}, v_{2,k}, \\
v_{4,k}, v_{3,k}, v_{4,k}, v_{1,k}, v_{12,k}, v_{9,k}, v_{12,k}, v_{14,k}, v_{13,k}, v_{14,k}, v_{18,k}, v_{15,k}, v_{16,k}, \\
v_{18,k}, v_{8,k}, v_{6,k}, v_{8,k}, v_{5,k}, v_{8,k}, v_{7,k}, v_{18,k}, v_{17,k})
\]

from \(v_{9,1}\) to \(v_{17,k}\) as shown in Figure 6.20. For clarity, the paths of \(\Pi'\) are dashed.
6.3.5 Conflict Resolution in the Minimum Spanning Tree

The goal of this section is to introduce some bypasses such that every vertex is adjacent to at most three edges from $T$. The PMCA-HPP$_2$ does this by considering every path $p$ in $P$ consisting only of edges from $T$ separately. If the vertex closest to $z = v_{9,1}$ in $p$ is internal to $p$, it is bypassed. Therefore, it bypasses $v_{4,i}$ between $v_{3,i}$ and $v_{1,i}$ and $v_{8,i}$ between $v_{5,i}$ and $v_{7,i}$, for $1 \leq i \leq k$, $v_{12,1}$ between $v_{11,1}$ and $v_{14,1}$, $v_{12,i}$ between $v_{9,i}$ and $v_{14,i}$, and between $v_{10,i}$ and $v_{4,i}$, for $2 \leq i \leq k$; $v_{18,i}$ between $v_{16,i}$ and $v_{8,i}$ and between $v_{17,i}$ and $v_{12,i+1}$, for $1 \leq i \leq k - 1$, and finally $v_{18,k}$ between $v_{16,k}$ and $v_{8,k}$. This results in the modified Eulerian path

$$P' = (v_{9,1}, v_{12,1}, v_{4,1}, v_{2,1}, v_{4,1}, v_{3,1}, v_{1,1}, v_{12,1}, v_{10,1}, v_{11,1}, v_{14,1}, v_{13,1}, v_{14,1}, v_{18,1}, v_{15,1}, v_{16,1}, v_{8,1}, v_{6,1}, v_{8,1}, v_{5,1}, v_{7,1}, v_{18,1}, v_{17,1}, v_{12,2}, v_{11,2}, v_{10,2}, v_{4,2}, v_{2,2}, v_{4,2}, v_{3,2}, v_{1,2}, v_{12,2}, v_{9,2}, v_{14,2}, v_{13,2}, v_{14,2}, v_{18,2}, v_{15,2}, v_{16,2}, v_{8,2}, v_{6,2}, v_{8,2}, v_{6,2}, v_{7,2}, v_{18,2}, v_{17,2}, \ldots; v_{12,2}, v_{11,2}, v_{10,2}, v_{4,2}, v_{2,2}, v_{4,2}, v_{3,2}, v_{1,2}, v_{12,2}, v_{9,2}, v_{14,2}, v_{13,2}, v_{14,2}, v_{18,2}, v_{15,2}, v_{16,2}, v_{8,2}, v_{6,2}, v_{8,2}, v_{6,2}, v_{7,2}, v_{18,2}, v_{17,2}, v_{9,2}, v_{14,2}, v_{13,2}, v_{14,2}, v_{18,2}, v_{15,2}, v_{16,2}, v_{8,2}, v_{6,2}, v_{8,2}, v_{6,2}, v_{7,2}, v_{18,2}, v_{17,2}, v_{9,2}, v_{14,2}, v_{13,2}, v_{14,2}, v_{18,2}, v_{15,2}, v_{16,2}, v_{8,2}, v_{6,2}, v_{8,2}, v_{6,2}, v_{7,2}, v_{18,2}, v_{17,2}, v_{9,2}, v_{14,2}, v_{13,2}, v_{14,2}, v_{18,2}, v_{15,2}, v_{16,2}, v_{8,2}, v_{6,2}, v_{8,2}, v_{6,2}, v_{7,2}, v_{18,2}, v_{17,2})$$

as shown in Figure 6.21.

Figure 6.20: The Eulerian path $P$. The paths of $\Pi'$ are dashed.

Figure 6.21: The modified Eulerian path $P'$. The paths of $\Pi'$ are dashed.
6.3.6 Conflict Resolution in the Modified Eulerian Path

The goal of the last step is to bypass vertices such that every vertex except \(w\) and \(z\) has degree 2, i.e., we have a Hamiltonian path from \(v_{0,1}\) to \(v_{17,k}\). The problematic vertices are thus all the vertices \(v_{k,i}, v_{8,i}, v_{12,i}, v_{14,i}, v_{18,i}\) for \(1 \leq i \leq k\). Because \(w = v_{17,k}\) is not a conflict in \(P'\), the PMCA-HPP2 starts with the resolution of an arbitrary conflict. It obtains the final Hamiltonian path

\[
P'' = (v_{9,1}, v_{12,1}, v_{2,1}, v_{4,1}, v_{3,1}, v_{1,1}, v_{10,1}, v_{11,1}, v_{13,1}, v_{14,1}, v_{18,1}, v_{15,1}, v_{16,1}, v_{6,1}, v_{8,1}, v_{5,1}, v_{7,1}, v_{17,1}, v_{12,2}, v_{11,2}, v_{10,2}, v_{2,2}, v_{4,2}, v_{3,2}, v_{1,2}, v_{9,2}, v_{13,2}, v_{14,2}, v_{18,2}, v_{15,2}, v_{16,2}, v_{6,2}, v_{8,2}, v_{5,2}, v_{7,2}, v_{17,2}, \ldots, v_{12,k}, v_{11,k}, v_{10,k}, v_{2,k}, v_{4,k}, v_{3,k}, v_{1,k}, v_{9,k}, v_{13,k}, v_{14,k}, v_{18,k}, v_{15,k}, v_{16,k}, v_{6,k}, v_{8,k}, v_{5,k}, v_{7,k}, v_{17,k})
\]

as shown in Figure 6.22 by doing the following:

- it bypasses \(v_{4,1}\) between \(v_{12,1}\) and \(v_{2,1}\), and every other \(v_{4,i}\) between \(v_{10,i}\) and \(v_{2,i}\);
- it bypasses \(v_{8,i}\) between \(v_{16,i}\) and \(v_{6,i}\), for \(1 \leq i \leq k\);
- it bypasses \(v_{12,1}\) between \(v_{1,1}\) and \(v_{10,1}\), and every other \(v_{12,i}\) between \(v_{1,i}\) and \(v_{9,i}\);
- it bypasses \(v_{14,1}\) between \(v_{11,1}\) and \(v_{13,1}\), and every other \(v_{14,i}\) between \(v_{9,i}\) and \(v_{13,i}\);
- it bypasses \(v_{18,i}\) between \(v_{7,i}\) and \(v_{17,i}\), for \(1 \leq i \leq k\).

![Figure 6.22: The Hamiltonian path \(P''\).](image)

Considering only the edges \(\{v_{1,i}, v_{9,i}\}, \{v_{2,i}, v_{10,i}\}, \{v_{6,i}, v_{16,i}\}, \{v_{7,i}, v_{17,i}\}\), and \(\{v_{9,i}, v_{13,i}\}\), for \(2 \leq i \leq k\), we obtain \(\text{cost}(P'') \geq 5(k - 1)\beta^2\). We have
6.3. The PMCA-HPP$_2$

thus shown that, for every $\beta \geq 1$ and arbitrarily small $\varepsilon > 0$, there exists an implementation $I$ of the PMCA-HPP$_2$ such that

$$\frac{\text{cost}(I(G_{18, k}(\beta)))}{\text{Opt}_{\Delta_\beta-\text{HPP}_2}(G_{18, k}(\beta))} \geq \frac{5(k-1)\beta^2}{3k + 2\beta^2 + 21\beta + 5 - \frac{2\beta^2 + \beta}{k}} \geq \frac{5}{3} \beta^2 - \varepsilon,$$

for sufficiently large $k$, i.e., we have shown that the upper bound of $\frac{5}{3} \beta^2$ on the approximation ratio of the PMCA-HPP$_2$ is tight.
Chapter 6. On the Approximation Ratio of the PMCA-HPP_t
Chapter 7

Additional Results

In this chapter, we first prove the tightness of an upper bound on an approximation algorithm for the metric HPP due to Forlizzi et al. [FHPS06]. Then we establish a lower bound of $4/3 - \varepsilon$ on a 1.4-approximation algorithm for metric TSP reoptimization due to Böckenhauer et al. [BFH+07]. Finally, we show the tightness of a 4/3-approximation algorithm for metric TSP reoptimization due to Berg and Hempel [BH09].

Reoptimization is the concept of finding a solution to an instance $I$ of an optimization problem when an optimal solution to another instance $I'$ is known, where $I'$ can be obtained from $I$ by a local modification. What this local modification is, is part of the problem definition. In graph-theoretic settings, often an edge length is changed or an edge or a vertex is added or removed. In Sections 7.2 and 7.3, we consider the following reoptimization problem.

**Definition 7.1.** The metric locally modified traveling salesman problem ($\Delta$-lm-TSP) is the following optimization problem.

**Input:** Two complete, weighted, metric graphs $G_1 = (V, E, c_1), G_2 = (V, E, c_2)$, where $c_1$ and $c_2$ differ only for one edge, and a shortest Hamiltonian cycle $H_1$ in $G_1$.

**Constraints:** $M(G_1, G_2, H_1) = \{(v_1, \ldots, v_n, v_1) \mid (v_1, \ldots, v_n) \text{ is a permutation of } V\}$ is the set of all Hamiltonian cycles in $G_2$.

**Costs:** The cost of a Hamiltonian cycle $H = (v_1, \ldots, v_n, v_1)$ is $c_2(H) = \sum_{i=1}^{n-1} c_2(v_i, v_{i+1}) + c_2(v_n, v_1)$.

**Goal:** Minimum.
7.1 Approximation of the Metric $HPP_2$

Consider the approximation algorithm for the metric $HPP_2$ shown in Procedure 7.1. It uses Procedure 7.2 that, in turn, is based on a result by Sekanina stating that the cube of every connected graph is Hamiltonian [Sek60].

**Theorem 7.2 [FHPS06].** Procedure 7.1 is a 3-approximation algorithm for $\Delta$-$HPP_2$. 

We show that this upper bound is tight, i.e., that the following holds.

**Theorem 7.3.** There exists an infinite family of graphs satisfying the triangle inequality on which Procedure 7.1 cannot achieve an approximation ratio of $3 - \varepsilon$, for any $\varepsilon > 0$.

**Proof.** Let $P_k$ be the complete graph with vertices $v_1, v_2, \ldots, v_k$, for some odd $k \geq 3$, with edge lengths $c(v_i, v_{i+1}) := 1$, for $1 \leq i \leq k - 1$, and maximum possible length for all other edges such that the triangle inequality is not violated. (See Figure 7.1, where only some edges are shown.) Let $s := v_1, t := v_k$.

![Figure 7.1: The graph $P_k$.](image)

**Procedure 7.1 T³-Metric-HPP² [FHPS06]**

**Input:** A complete weighted metric graph $G = (V, E, c)$ and two vertices $s, t \in V$.

1. Find a minimum spanning tree $T$ in $G$ containing the edge $\{s, t\}$.
2. Call HCT³ with inputs $T$ and $\{s, t\}$ to obtain a Hamiltonian cycle $H$ in $T^3$ containing $\{s, t\}$.
3. Remove $\{s, t\}$ from $H$ to obtain a Hamiltonian path $P$ from $s$ to $t$ in $G$.

**Output:** $P$.

---

1[FHPS06] incorrectly references [AB95] instead of [And01] as the source of the HCT³ algorithm.
7.1. Approximation of the Metric HPP

Procedure 7.2 HCT\(^3\) [And01]

**Input:** A tree \(T\) with \(|E(T)| \geq 2\) and an edge \(e = \{u_1, u_2\}\) in \(T\).

1. Let \(T_i\) be the component of \(T - e\) containing \(u_i\), for \(i = 1, 2\).
2. for \(i = 1, 2\) do
3.  if \(|E(T_i)| \geq 1\) then
4.      pick \(u'_i \in V(T_i)\) such that \(\{u_i, u'_i\} \in E(T_i)\)
5.      \(e_i := \{u_i, u'_i\}\)
6.  else
7.      \(u'_i := u_i\)
8.  end
9. if \(|E(T_i)| \geq 2\) then
10. recursively call HCT\(^3\) with inputs \(T_i\) and \(e_i\) to obtain a Hamiltonian cycle \(H_i\) in \(T^3_i\) containing \(e_i\)
11. \(P_i := H_i - e_i\)
12. else
13. \(P_i := T_i\)
14. end
15. end
16. \((P_i\) is a path from \(u_i\) to \(u'_i\), for \(i = 1, 2\).)
17. Put together \(e, P_1, P_2,\) and \(\{u'_1, u'_2\}\) to obtain a Hamiltonian cycle \(H\) in \(T^3\) containing \(e\).

**Output:** \(H\).

It is easy to see that the optimal solution has length \(k - 1\) and consists of all the edges shown in Figure 7.1, i.e., \(P_{Opt} = (v_1, v_2, \ldots, v_k)\). We now show one possible implementation of the T\(^3\)-Metric-HPP\(_2\) that, on input \(P_k\), returns a Hamiltonian path of length \(3k - 7\) from \(v_1\) to \(v_k\).

In the first step, Procedure 7.1 computes a minimum spanning tree containing the edge \(\{s, t\}\). Such a minimum spanning tree always contains all edges of length 1 except one as well as the edge \(\{s, t\} = \{v_1, v_k\}\). Figure 7.2 shows the minimum spanning tree \(T\) that does not contain the edge \(\{v_{k-1}, v_k\}\).

![Figure 7.2: A minimum spanning tree in \(P_k\).](image)

In the second step, Procedure 7.2 is called with inputs \(T\) and \(\{v_1, v_k\}\). Let
\( u_1 = v_1, u_2 = v_k \). Therefore, \( T_1 \) is the tree with the vertices \( v_1, v_2, \ldots, v_{k-1} \) and edges \{\( v_i, v_{i+1} \}\), for \( 1 \leq i \leq k-2 \), and \( T_2 \) is the tree with the vertex \( v_k \). For \( i = 2 \), Procedure 7.2 sets \( u'_2 := u_2 = v_k \) and \( P_2 := T_2 = (\{v_k\}, \emptyset) \).

Let us now see what happens for \( i = 1 \).

Because \( T_1 \) indeed contains an edge, Procedure 7.2 sets \( u'_1 := v_2 \) and \( e_1 := \{u_1, u'_1\} = \{v_1, v_2\} \). We know that \( k \geq 3 \), so there are at least two edges in \( T_1 \). Hence, Procedure 7.2 recursively calls itself with inputs \( T_1 \) and \( e_1 \).

Using the same argument, one can see that this, in turn, results in another recursive call with inputs \((\{v_2, v_3, \ldots, v_{k-1}\}, \{\{v_i, v_{i+1}\} \mid 2 \leq i \leq k-2\})\), i.e., the path from \( v_2 \) to \( v_{k-1} \), and the edge \( \{v_2, v_3\} \). With every recursive call, the leftmost vertex, i.e., the vertex with lowest index in the tree, is “dropped” and only the right side is considered. This stops when the algorithm reaches \( v_{k-3} \), as we shall see now.

Consider the moment when Procedure 7.2 is called with the following inputs. First, the path from \( v_{k-3} \) to \( v_{k-1} \) and second the edge \( \{v_{k-3}, v_{k-1}\} \). It will “split” the path once again and call itself recursively for the path from \( v_{k-2} \) to \( v_{k-1} \).

This last execution is different. The algorithm does set \( u'_2 := v_{k-1} \) and \( e_2 := \{v_{k-2}, v_{k-1}\} \). But it does not recursively call itself, because the remaining graph contains only one edge. Instead, it just sets \( P_2 := T_2 \), i.e., the single edge \( e_2 \).

Combining \( e = \{v_{k-3}, v_{k-2}\}, P_2 := T_2 = \{v_{k-2}, v_{k-1}\}, \{u'_1, u'_2\} = \{v_{k-3}, v_{k-1}\} \), and \( P_1 = (v_{k-3}) \), we obtain the Hamiltonian cycle \( H_3 = (v_{k-3}, v_{k-2}, v_{k-1}, v_k) \) for the vertices \( v_{k-3}, v_{k-2}, v_{k-1} \).

Now we go back up one step in the recursion. Remember that \( H_3 \) was created when \( \text{HCT}^3 \) called itself for the vertices \( \{v_{k-3}, v_{k-2}, v_{k-1}\} \). When this call returns, \( \text{HCT}^3 \) removes the edge \( \{v_{k-3}, v_{k-2}\} \) to obtain the Hamiltonian path \( P_2 = (v_{k-2}, v_{k-1}, v_k) \). Furthermore, in the current recursion step, \( e = \{v_{k-4}, v_{k-3}\}, P_1 = (v_{k-4}), \) and \( \{u'_1, u'_2\} = \{v_{k-4}, v_{k-2}\} \). The algorithm combines \( P_1, \{u'_1, u'_2\}, P_2, \) and \( e \) to obtain the Hamiltonian cycle \( (v_{k-4}, v_{k-2}, v_{k-1}, v_k) \) for the vertices \( v_{k-4} \) to \( v_k \) (see Figure 7.3).

**Figure 7.3:** The situation after the construction of \( H_4 \).

The argument is exactly the same for every recursion step, and we obtain the Hamiltonian cycle

\[
(v_1, v_3, v_5, \ldots, v_{k-2}, v_{k-1}, v_k, v_{k-3}, \ldots, v_2, v_1)
\]
as depicted in Figure 7.4 for the vertices \( v_1 \) to \( v_{k-1} \).
7.1. Approximation of the Metric HPP

\[ s = v_1, v_2, v_3, \ldots, v_{k-4}, v_{k-3}, v_{k-2}, v_{k-1}, v_k = t \]

Figure 7.4: The situation after the construction of \( H_{k-1} \).

After this step, the algorithm removes the edge \( \{u_1, u'_1\} = \{v_1, v_2\} \) to obtain the Hamiltonian path \( P_1 \). Then it combines \( P_1, P_2 = (v_k), e = \{v_1, v_k\} \), and \( \{u'_1, u'_2\} = \{v_2, v_k\} \) to obtain the final Hamiltonian cycle

\[ H = (v_1, v_3, v_5, \ldots, v_{k-2}, v_{k-1}, v_{k-3}, \ldots, v_2, v_k, v_1) \]

This concludes the execution of the HCT\(^3\) algorithm, i.e., of step 2 of Procedure 7.1. All that is left to do is remove the edge \( \{v_1, v_k\} \) from \( H \) to obtain the final Hamiltonian path

\[ P = (v_1, v_3, v_5, \ldots, v_{k-2}, v_{k-1}, v_{k-3}, \ldots, v_2, v_k) \]

as shown in Figure 7.5.

\[ s = v_1, v_2, v_3, \ldots, v_{k-4}, v_{k-3}, v_{k-2}, v_{k-1}, v_k = t \]

Figure 7.5: The final Hamiltonian path \( P \).

This path contains \( (k - 3)/2 \) edges \( \{v_{2i+1}, v_{2i+3}\} \), for \( 0 \leq i \leq (k - 5)/2 \), of length 2 each, as well as another \( (k - 3)/2 \) edges \( \{v_{2i}, v_{2i+2}\} \), for \( 1 \leq i \leq (k - 3)/2 \), also of length 2 each. Additionally, we have the edge \( \{v_{k-2}, v_{k-1}\} \) of length 1 and the edge \( \{v_2, v_k\} \) of length \( k - 2 \). In total, the length of \( P \) amounts to \( 2 \cdot (k - 3)/2 \cdot 2 + 1 + k - 2 = 2 \cdot (k - 3) + k - 1 = 3k - 7 \).

We have thus shown that, for arbitrarily small \( \varepsilon > 0 \), there exists an implementation \( I \) of the T\(^3\)-Metric-HPP\(_2\) and the HCT\(^3\) such that

\[ \frac{\text{cost}(I(P_k))}{\text{Opt}_{\Delta-HPP_2}(P_k)} \geq \frac{3k - 7}{k - 1} > 3 - \varepsilon, \]

for sufficiently large \( k \), i.e., we have shown that the upper bound of 3 on the approximation ratio of the T\(^3\)-Metric-HPP\(_2\) is tight.

Observe that the only nondeterministic step of Procedure 7.1 on inputs \( P_k, v_1, v_k \) is the construction of the minimum spanning tree in step 1. The end result, however, always has asymptotic length \( 3k \); independent of the minimum spanning tree. This means that every possible implementation of the T\(^3\)-Metric-HPP\(_2\) cannot achieve an approximation ratio better than 3 for these inputs.
7.2 1.4-Approximation of the Δ-LM-TSP

Procedure 7.3 shows a reoptimization algorithm devised by Böckenhauer et al. [BFH+07] for the Δ-LM-TSP with increased edge cost, i.e., \( c_2(e) > c_1(e) \), using Hoogeveen’s HPP\(_2\) approximation algorithm (Procedure 7.4).\(^2\)

**Procedure 7.3** Δ-LM-TSP Approximation Algorithm [BFH+07]

**Input:** Two complete weighted metric graphs \( G_1 = (V, E, c_1), G_2 = (V, E, c_2) \), where \( c_1 \) and \( c_2 \) differ only for one edge \( e \) and \( c_2(e) > c_1(e) \), and a shortest Hamiltonian cycle \( H_1 \) in \( G_1 \).

1: \( H := H_1 \)

2: arbitrarily choose \( v \) to be one of the two vertices of \( e \)

3: for every pair \( e', e'' \) of edges incident to \( v \), where \( e' \neq e, e'' \neq e \) do

4: \( v' := e' - v, v'' := e'' - v \)

5: use Hoogeveen’s algorithm to compute a Hamiltonian path \( P \) from \( v' \) to \( v'' \) in \( G_2[V - v] \)

6: obtain \( H' \) by concatenating \( e', e'' \), and \( P \)

7: if \( c(H') < c(H) \) then

8: \( H := H' \)

9: end

10: end

**Output:** \( H \).

**Procedure 7.4** Hoogeveen’s Algorithm [Hoo91]

**Input:** A complete weighted metric graph \( G = (V, E, c) \), and two vertices \( u, v \in V \).

Construct a minimum spanning tree \( T \) for \( G \).

Determine the set \( S \) consisting of fixed endpoints of even degree and other vertices of odd degree.

Construct a minimum matching \( M \) for \( S \).

Find an Eulerian path \( E \) in the graph \( T \cup M \).

Transform \( E \) into a Hamiltonian path \( P \).

**Output:** \( P \).

**Theorem 7.4** [BFH+07]. Procedure 7.3 is a 1.4-approximation algorithm for the Δ-LM-TSP with \( c_2(e) > c_1(e) \). \( \square \)

\(^2\)[BFH+07] does not explicitly state the condition \( e' \neq e, e'' \neq e \) from line 3. It is, however, clear that a Hamiltonian cycle \( H' \) containing \( e \) can never be shorter than \( H_1 \). The purpose of this condition is to simplify the proof of Lemma 7.5.
We now show a lower bound of 4/3 for Procedure 7.3 for the case where the edge cost is increased, i.e., we show that the following holds.

**Lemma 7.5.** There exists an infinite family of inputs for the $\Delta$-LM-TSP, with $c_2(e) > c_1(e)$, on which Procedure 7.3 cannot achieve an approximation ratio of $4/3 - \varepsilon$, for any $\varepsilon > 0$.

Let $G_{6,k}$ be the complete graph with the vertices $\{u_1, u_2, u_3, u_4, u_5, u_6\} \cup \{v_i \mid 1 \leq i \leq k - 1\}$, for $k \in \mathbb{N}$, with edge lengths
\[
\begin{align*}
    c(u_1, u_2) &= c(u_1, u_5) = c(u_2, u_5) = c(u_2, u_3) = 1, \\
    c(u_3, u_4) &= c(u_3, u_6) = c(u_4, u_6) = c(u_5, u_6) := 1, \\
    c(u_1, v_1) &= c(v_i, v_{i+1}) = c(v_{k-1}, u_4) := 1/k,
\end{align*}
\]
for $1 \leq i \leq k - 2$, and maximum possible length for all other edges such that the triangle inequality is not violated.

Figure 7.6 shows the basic structure of the graph. (Only some edges are shown.)

![Graph G_{6,k}](https://example.com/graph.png)

**Figure 7.6:** The graph $G_{6,k}$.

**Lemma 7.6.** The graph $G_{6,k}$ satisfies the triangle inequality.

**Proof.** Let $\{u, v, w\}$ be a triangle, and let $e = \{u, v\}$ be the longest edge in it, i.e., the one that could be responsible for a violation. We show that there always exists another edge in this triangle that is long enough such that $e$ does not violate the $\beta$-triangle inequality.

The edge $e$ can, by construction, only be one of those edges for which we explicitly defined the edge length. Furthermore, all edges for which we did not explicitly define the length have length at least $2/k$. All edges in the entire graph thus have length at least $1/k$, and we have to prove the statement only for the edges of length 1.

Assume that $e = \{u_1, u_2\}$. Because every vertex has distance at least 1 from $u_2$, the edge $\{w, u_2\}$ has length at least 1. Therefore, $c(u_1, u_2) = 1 \leq 1/k + 1 \leq c(u_1, w) + c(w, u_2)$. The proof works analogously for all the other edges $\{u_1, u_5\}, \{u_2, u_3\}, \{u_2, u_5\}, \{u_3, u_4\}, \{u_3, u_6\}, \{u_4, u_6\}, \{u_5, u_6\}$. \qed
Chapter 7. Additional Results

Figure 7.7: The Hamiltonian cycle $H_{\text{Opt}}$.

Lemma 7.7. The Hamiltonian cycle $H_{\text{Opt}} := (u_1, u_5, u_2, u_3, u_6, u_4, v_{k-1}, v_{k-2}, \ldots, v_1, u_1)$ of length 6 as shown in Figure 7.7 is optimal.

Proof. If a Hamiltonian cycle for $G_{6,k}$ contains the vertices $u_2, u_3, u_5, u_6$ in consecutive (but arbitrary) order, then it contains at least five edges of length 1 (three to connect them to each other and two to connect them to the rest of the graph). If this is not the case, then it contains at least six edges of length at least 1. The shortest edges in the entire graph have length $1/k$, and there are $k+5$ vertices in the graph, so the shortest Hamiltonian cycle has length $5 \cdot 1 + (k+5-5) \cdot 1/k = 6$.

Let $G'_{6,k}$ be the graph $G_{6,k}$ with the only change that the edge $\{u_2, u_3\}$ has length 3 instead of 1, i.e., $c(u_2, u_3) := 3$.

Lemma 7.8. The graph $G'_{6,k}$ satisfies the triangle inequality.

Proof. Let $e$ be an edge that could be responsible for a violation of the triangle inequality, as in the previous proof. The only possibility is $e = \{u_2, u_3\}$, all other edges are covered in the previous proof.

If $w \in \{u_1, u_5\}$, then $c(u_2, u_3) = 1 \leq 3 = 1 + 2 = c(u_2, w) + c(w, u_3)$. If $w \in \{u_4, u_6\}$, then $c(u_2, u_3) = 1 \leq 3 = 2 + 1 + c(u_2, w) + c(w, u_3)$. If $w = v_i$, for some $1 \leq i \leq k-1$, then $c(u_2, u_3) = 1 \leq 3 = (1 + i/k) + (1 + 1 - i/k) = c(u_2, w) + c(w, u_3)$.

The optimal Hamiltonian cycle for $G'_{6,k}$, $H'_{\text{Opt}} := (u_1, u_2, u_5, u_6, u_3, u_4, v_{k-1}, v_{k-2}, \ldots, v_1, u_1)$, can be seen in Figure 7.8.

We now show one possible implementation of Procedure 7.3 that, on inputs $G_{6,k}, G'_{6,k}, H_{\text{Opt}}$, returns a Hamiltonian cycle of length 8 in $G'_{6,k}$.

Lemma 7.9. Let $V := V(G'_{6,k})$. A tree of overall length 4 in the graph $G'_{6,k}[V - u_2]$ is a minimum spanning tree.

---

The optimality immediately follows from the fact that $H'_{\text{Opt}}$ has the same length as $H_{\text{Opt}}$ and from Lemma 7.7.
7.2. 1.4-Approximation of the $\Delta$-LM-TSP

![Figure 7.8: The Hamiltonian cycle $H'_\text{opt}$.](image)

**Proof.** Since the edges of length $1/k$ are the shortest in the entire graph, the path $(u_1, v_1, v_2, \ldots, v_{k-1}, u_4)$ of length exactly 1 clearly is a minimum spanning tree for the respective vertices. Then, the vertices $u_3, u_5,$ and $u_6$ still have to be connected to this path. Every such connection has length at least 1.

Let $v := u_2.$ The essential part of Procedure 7.3 is the for-loop that uses Hoogeveen’s algorithm to compute Hamiltonian paths between pairs of fixed endpoints. All we have to do is go through all these pairs.

**Case 1:** $e', e'' = \{u_2, u_1\}, \{u_2, u_4\}.$ Procedure 7.4 computes the minimum spanning tree with all edges of length $1/k$ as well as the edges $\{u_1, u_5\}, \{u_3, u_6\}, \{u_4, u_6\}.$ Then, it computes the minimum matching $M = \{\{u_1, u_5\}, \{u_3, u_6\}\}$ for the set $S = \{u_1, u_3, u_5, u_6\}.$ Afterwards, it computes the Eulerian path $E = (u_1, u_5, u_1, v_1, v_2, \ldots, v_{k-1}, u_4, u_6, u_3, u_4)$ shown in Figure 7.9.1 and transforms it into the Hamiltonian path $P = (u_1, u_5, v_1, v_2, \ldots, v_{k-1}, u_6, u_3, u_4)$ shown in Figure 7.9.2. Procedure 7.3 the combines $P, e',$ and $e''$ to obtain the Hamiltonian cycle $H = (u_2, u_1, u_5, v_1, v_2, \ldots, v_{k-1}, u_6, u_3, u_4, u_2)$ of length 9.

**Case 2:** $e', e'' = \{u_2, u_1\}, \{u_2, u_3\}.$ Procedure 7.4 computes the minimum spanning tree with all edges of length $1/k$ as well as the edges $\{u_1, u_5\}, \{u_3, u_4\}, \{u_5, u_6\}.$ Then, it computes the minimum matching $M = \{\{u_1, u_5\}, \{u_3, u_6\}\}$ for the set $S = \{u_1, u_3, u_5, u_6\}.$ Afterwards, it computes the Eulerian path $E = (u_5, u_1, v_1, v_2, \ldots, v_{k-1}, u_4, u_6, u_5, u_1)$ shown in Figure 7.9.3 and transforms it into the Hamiltonian path $P = (u_5, v_1, v_2, \ldots, v_{k-1}, u_4, u_6, u_3, u_4, u_2)$ shown in Figure 7.9.4. Procedure 7.3 the combines $P, e',$ and $e''$ to obtain the Hamiltonian cycle $H = (u_2, u_5, v_1, v_2, \ldots, v_{k-1}, u_4, u_3, u_6, u_1, u_2)$ of length 8.

**Case 3:** $e', e'' = \{u_2, u_1\}, \{u_2, u_6\}.$ Procedure 7.4 computes the minimum spanning tree with all edges of length $1/k$ as well as the edges $\{u_1, u_5\}, \{u_3, u_4\}, \{u_5, u_6\}.$ Then, it computes the minimum matching $M = \{\{u_1, u_3\}\}$ for the set $S = \{u_1, u_3\}.$ Afterwards, it computes the Eulerian path $E = (u_6, u_5, u_1, v_1, v_2, \ldots, v_{k-1}, u_4, u_3, u_4)$ shown in Figure 7.9.5 and trans-
forms it into the Hamiltonian path \( P = (u_6, u_5, v_1, v_2, \ldots, v_{k-1}, u_4, u_3, u_1) \) shown in Figure 7.9.6. Procedure 7.3 the combines \( P, e' \), and \( e'' \) to obtain the Hamiltonian cycle \( H = (u_2, u_6, u_5, v_1, v_2, \ldots, v_{k-1}, u_4, u_3, u_1, u_2) \) of length 9.

**Case 4:** \( e', e'' = \{u_2, u_1\}, \{u_2, v_1\} \), for \( 1 \leq i \leq k - 1 \). Procedure 7.4 computes the minimum spanning tree with all edges of length 1/k as well as the edges \( \{u_1, u_2\}, \{u_3, u_6\}, \{u_4, u_6\} \). Because the minimum matching depends on \( i \), we have to make another case distinction.

If \( i \leq \lceil \frac{k}{2} \rceil \), it computes the minimum matching \( M = \{(u_1, v_1), (u_3, u_5)\} \) for the set \( S = \{u_1, u_3, u_5, v_1\} \). Then, it computes the Eulerian path \( E = (v_1, u_1, u_5, u_3, u_6, u_4, v_{k-1}, v_{k-2}, \ldots, v_1, u_1) \) shown in Figure 7.9.7 and transforms it into the Hamiltonian path \( P = (v_1, u_5, u_3, u_6, v_{k-1}, v_{k-2}, \ldots, v_{i+1}, v_{i-1}, \ldots, v_1, u_1) \) shown in Figure 7.9.8). Procedure 7.3 then combines \( P, e' \), and \( e'' \) to obtain the Hamiltonian cycle \( H = (u_2, v_1, u_5, u_3, u_6, u_4, v_{k-1}, v_{k-2}, \ldots, v_{i+1}, v_{i-1}, \ldots, v_1, u_1, u_2) \) of length \( 8 + 2i/k \geq 8 \).

If \( i > \lceil \frac{k}{2} \rceil \), it computes the minimum matching \( M = \{(u_1, u_5), (u_3, v_1)\} \) for the set \( S = \{u_1, u_3, u_5, v_1\} \). Then, it computes the Eulerian path \( E = (v_1, u_3, u_6, u_4, v_{k-1}, v_{k-2}, \ldots, v_1, u_5, u_1) \) shown in Figure 7.9.9 and transforms it into the Hamiltonian path \( P = (v_1, u_3, u_6, v_4, v_{k-1}, v_{k-2}, \ldots, v_{i+1}, v_{i-1}, \ldots, u_5, u_1) \) shown in Figure 7.9.10. Procedure 7.3 then combines \( P, e' \), and \( e'' \) to obtain the Hamiltonian cycle \( H = (u_2, v_1, u_3, u_6, v_4, v_{k-1}, v_{k-2}, \ldots, v_{i+1}, v_{i-1}, \ldots, v_1, u_5, u_1, u_2) \) of length 9.

**Case 5:** \( e', e'' = \{u_2, u_4\}, \{u_2, u_5\} \). Procedure 7.4 computes the minimum spanning tree with all edges of length 1/k as well as the edges \( \{u_1, u_5\}, \{u_3, u_4\}, \{u_3, u_6\} \). Then, it computes the minimum matching \( M = \{(u_3, u_4), (u_5, u_6)\} \) for the set \( S = \{u_3, u_4, u_5, u_6\} \). Afterwards, it computes the Eulerian path \( E = (u_5, u_6, u_5, u_1, v_1, v_2, \ldots, v_{k-1}, u_4, u_3, u_4) \) shown in Figure 7.9.11 and transforms it into the Hamiltonian path \( P = (u_5, u_6, u_1, v_1, v_2, \ldots, v_{k-1}, u_3, u_4) \) shown in Figure 7.9.12. Procedure 7.3 the combines \( P, e' \), and \( e'' \) to obtain the Hamiltonian cycle \( H = (u_2, u_5, u_6, u_1, v_1, v_2, \ldots, v_{k-1}, u_3, u_4, u_2) \) of length 9.

**Case 6:** \( e', e'' = \{u_2, u_4\}, \{u_2, u_6\} \). Procedure 7.4 computes the minimum spanning tree with all edges of length 1/k as well as the edges \( \{u_1, u_5\}, \{u_3, u_4\}, \{u_3, u_6\} \). Then, it computes the minimum matching \( M = \{(u_4, u_5)\} \) for the set \( S = \{u_4, u_5\} \). Afterwards, it computes the Eulerian path \( E = (u_4, u_5, u_1, v_1, v_2, \ldots, v_{k-1}, u_4, u_3, u_6) \) shown in Figure 7.9.13 and transforms it into the Hamiltonian path \( P = (u_4, u_5, u_1, v_1, v_2, \ldots, v_{k-1}, u_3, u_6) \) shown in Figure 7.9.14. Procedure 7.3 the combines \( P, e' \), and \( e'' \) to obtain the Hamiltonian cycle \( H = (u_2, u_4, u_5, u_1, v_1, v_2, \ldots, v_{k-1}, u_3, u_6, u_2) \) of length 10.

**Case 7:** \( e', e'' = \{u_2, u_4\}, \{u_2, v_3\} \), for \( 1 \leq i \leq k - 1 \). Procedure 7.4 computes the minimum spanning tree with all edges of length 1/k as well as
the edges \{u_1, u_5\}, \{u_3, u_6\}, \{u_4, u_6\}.

If \(i \leq \left\lfloor \frac{k}{2} \right\rfloor\), it computes the minimum matching \(M = \{\{u_3, u_4\}, \{u_5, v_i\}\}\) for the set \(S = \{u_3, u_4, u_5, v_i\}\). Then, it computes the Eulerian path \(E = (v_i, u_5, u_1, v_1, v_2, \ldots, v_{k-1}, u_4, u_3, u_6, u_4)\) shown in Figure 7.9.15 and transforms it into the Hamiltonian path \(P = (v_i, u_5, u_1, v_1, v_2, \ldots, v_{i-1}, v_{i+1}, \ldots, v_{k-1}, u_3, u_6, u_4)\) shown in Figure 7.9.16. Procedure 7.3 then combines \(P, e', e''\) to obtain the Hamiltonian cycle \(H = (u_2, v_i, u_5, u_1, v_1, v_2, \ldots, v_{i-1}, v_{i+1}, \ldots, v_{k-1}, u_3, u_6, u_4, u_2)\) of length \(9 + 2i/k \geq 9\).

If \(i > \left\lceil \frac{k}{2} \right\rceil\), it computes the minimum matching \(M = \{\{u_3, u_5\}, \{u_4, v_i\}\}\) for the set \(S = \{u_3, u_4, u_5, v_i\}\). Then, it computes the Eulerian path \(E = (v_i, u_4, u_6, u_3, u_5, u_1, v_1, v_2, \ldots, v_{k-1}, u_4)\) shown in Figure 7.9.17 and transforms it into the Hamiltonian path \(P = (v_i, u_6, u_3, u_5, u_1, v_1, v_2, \ldots, v_{i-1}, v_{i+1}, \ldots, v_{k-1}, u_4)\) shown in Figure 7.9.18. Procedure 7.3 then combines \(P, e', e''\) to obtain the Hamiltonian cycle \(H = (u_2, v_i, u_6, u_3, u_5, u_1, v_1, v_2, \ldots, v_{i-1}, v_{i+1}, \ldots, v_{k-1}, u_3, u_4, u_2)\) of length 10.

Case 8: \(e', e'' = \{u_2, u_3\}, \{u_2, u_6\}\). Procedure 7.4 computes the minimum spanning tree with all edges of length \(1/k\) as well as the edges \{u_1, u_5\}, \{u_3, u_6\}, \{u_5, u_6\}. Then, it computes the minimum matching \(M = \{\{u_3, u_4\}, \{u_5, u_6\}\}\) for the set \(S = \{u_3, u_4, u_5, u_6\}\). Afterwards, it computes the Eulerian path \(E = (u_5, u_6, u_3, u_4, v_{k-1}, v_{k-2}, \ldots, v_1, u_1, v_5)\) shown in Figure 7.9.19 and transforms it into the Hamiltonian path \(P = (u_5, u_3, u_4, v_{k-1}, v_{k-2}, \ldots, v_1, u_1, u_6)\) shown in Figure 7.9.20. Procedure 7.3 then combines \(P, e', e''\) to obtain the Hamiltonian cycle \(H = (u_2, u_3, u_4, v_{k-1}, v_{k-2}, \ldots, v_1, u_1, u_6, u_2)\) of length 9.

Case 9: \(e', e'' = \{u_2, u_5\}, \{u_2, v_i\}, 1 \leq i \leq k-1\). Procedure 7.4 computes the minimum spanning tree with all edges of length \(1/k\) as well as the edges \{u_1, u_5\}, \{u_3, u_6\}, \{u_5, u_6\}.

If \(i \leq \left\lfloor \frac{k}{2} \right\rfloor\), it computes the minimum matching \(M = \{\{u_3, u_4\}, \{u_5, v_i\}\}\) for the set \(S = \{u_3, u_4, u_5, v_i\}\). Then, it computes the Eulerian path \(E = (v_i, u_5, u_6, u_3, u_4, v_{k-1}, v_{k-2}, \ldots, v_1, u_1, u_5)\) shown in Figure 7.9.21 and transforms it into the Hamiltonian path \(P = (v_i, u_6, u_3, u_4, v_{k-1}, v_{k-2}, \ldots, v_{i-1}, v_{i+1}, v_i, \ldots, v_1, u_1, u_5)\) shown in Figure 7.9.22. Procedure 7.3 then combines \(P, e', e''\) to obtain the Hamiltonian cycle \(H = (u_2, u_1, u_6, u_3, u_4, v_{k-1}, v_{k-2}, \ldots, v_i, \ldots, v_1, u_1, u_5, u_2)\) of length 8.

If \(i > \left\lceil \frac{k}{2} \right\rceil\), it computes the minimum matching \(M = \{\{u_3, u_5\}, \{u_4, v_i\}\}\) for the set \(S = \{u_3, u_4, u_5, v_i\}\). Then, it computes the Eulerian path \(E = (v_i, u_4, v_{k-1}, v_{k-2}, \ldots, v_1, u_1, u_5, u_3, u_6, u_5)\) shown in Figure 7.9.23 and transforms it into the Hamiltonian path \(P = (v_i, u_4, v_{k-1}, v_{k-2}, \ldots, v_{i-1}, v_{i+1}, v_i, v_1, u_1, u_5, u_6, u_5)\) shown in Figure 7.9.24. Procedure 7.3 then combines \(P, e', e''\) to obtain the Hamiltonian cycle \(H = (u_2, v_i, u_4, v_{k-1}, v_{k-2}, \ldots, v_{i+1}, v_{i-1}, \ldots, v_1, u_1, u_3, u_6, u_5, u_2)\) of length 8.
Chapter 7. Additional Results

Case 10: $e', e'' = \{u_2, u_6\}, \{u_2, v_i\}$, for $1 \leq i \leq k - 1$. Procedure 7.4 computes the minimum spanning tree with all edges of length $1/k$ as well as the edges $\{u_1, u_5\}, \{u_3, u_4\}, \{u_4, u_6\}$.

If $i \leq \lfloor \frac{k}{2} \rfloor$, it computes the minimum matching $M = \{\{u_3, u_4\}, \{u_5, v_i\}\}$ for the set $S = \{u_3, u_4, u_5, v_i\}$. Then, it computes the Eulerian path $E = (v_i, u_5, u_1, v_1, v_2, \ldots, v_{k-1}, u_4, u_3, u_4, u_6)$ shown in Figure 7.9.25 and transforms it into the Hamiltonian path $P = (v_i, u_5, u_1, v_1, v_2, \ldots, v_{i-1}, v_{i+1}, \ldots, v_{k-1}, u_3, u_4, u_6)$ shown in Figure 7.9.26. Procedure 7.3 then combines $P, e'$, and $e''$ to obtain the Hamiltonian cycle $H = (u_2, v_i, u_5, u_1, v_1, v_2, \ldots, v_{i-1}, v_{i+1}, \ldots, v_{k-1}, u_3, u_4, u_6, u_2)$ of length $9 + 2i/k \geq 9$.

If $i > \lceil \frac{k}{2} \rceil$, it computes the minimum matching $M = \{\{u_3, u_5\}, \{u_4, v_i\}\}$ for the set $S = \{u_3, u_4, u_5, v_i\}$. Then, it computes the Eulerian path $E = (v_i, u_4, u_3, u_5, u_1, v_1, v_2, \ldots, v_{k-1}, u_4, u_6)$ shown in Figure 7.9.27 and transforms it into the Hamiltonian path $P = (v_i, u_4, u_3, u_5, u_1, v_1, v_2, \ldots, v_{i-1}, v_{i+1}, \ldots, v_{k-1}, u_4)$ shown in Figure 7.9.28. Procedure 7.3 then combines $P, e'$, and $e''$ to obtain the Hamiltonian cycle $H = (u_2, v_i, u_4, u_3, u_5, u_1, v_1, v_2, \ldots, v_{i-1}, v_{i+1}, \ldots, v_{k-1}, u_3, u_4, u_2)$ of length 10.

**Figure 7.9:** The case distinction needed in the proof of Lemma 7.5. In the Eulerian paths, the edges of $M$ are dashed and the vertices in $S$ are circled.
Figure 7.9: The case distinction needed in the proof of Lemma 7.5. In the Eulerian paths, the edges of $M$ are dashed and the vertices in $S$ are circled.
Chapter 7. Additional Results

Figure 7.9: The case distinction needed in the proof of Lemma 7.5. In the Eulerian paths, the edges of $M$ are dashed and the vertices in $S$ are circled.
7.3 4/3-Approximation of the $\Delta$-LM-TSP

Berg and Hempel [BH09] presented the reoptimization algorithm for $\Delta$-LM-TSP shown in Procedure 7.5.  

Procedure 7.5 $\Delta$-LM-TSP Approximation Algorithm [BH09]

**Input:** Two complete weighted metric graphs $G_1 = (V, E, c_1), G_2 = (V, E, c_2)$, where $c_1$ and $c_2$ differ only for one edge $e$ where $c_2(e) > c_1(e)$, and a shortest Hamiltonian cycle $H_1$ in $G_1$.

1. Apply the Christofides algorithm on $G_2$ to obtain a Hamiltonian cycle $H_2$.
2. Let $H$ be the shorter of the two Hamiltonian cycles $H_1, H_2$.

**Output:** $H$.

They proved that this algorithm has a better approximation ratio than the one presented in the previous section.

**Theorem 7.10** [BH09]. Procedure 7.5 is a 4/3-approximation algorithm for the $\Delta$-LM-TSP.  

---

![Figure 7.9](image-url)  

**Figure 7.9:** The case distinction needed in the proof of Lemma 7.5. In the Eulerian paths, the edges of $M$ are dashed and the vertices in $S$ are circled.

In none of these cases, a Hamiltonian cycle of length shorter than 8 is obtained and the Hamiltonian cycle $H_{\text{Opt}}$ also has length 8 in $G'_{6,k}$. We have thus shown that, for arbitrarily small $\varepsilon > 0$, there exists an implementation $I$ of Procedure 7.3 such that

$$\frac{\text{cost}(I(G_{6,k}, G'_{6,k}, H_{\text{Opt}}))}{\text{Opt}_{\Delta\text{-LM-TSP}}(G_{6,k}, G'_{6,k}, H_{\text{Opt}})} \geq \frac{8}{6} > \frac{4}{3} - \varepsilon,$$

i.e., we have established a lower bound of $4/3 - \varepsilon$ on the approximation ratio of Procedure 7.3 if $c_2(e) > c_1(e)$. 

---
We show that this upper bound is tight in the case of an increased edge cost, i.e., we prove the following result.

**Lemma 7.11.** There exists an infinite family of inputs for the Δ-LM-TSP, with \( c_2(e) > c_1(e) \), on which Procedure 7.5 cannot achieve an approximation ratio of \( 4/3 - \varepsilon \), for any \( \varepsilon > 0 \).

Let \( G_{4,k} \) be the complete graph with the vertices \( \{a_i, b_i, c_i, d_i \mid 1 \leq i \leq k\} \), for \( k \in \mathbb{N}, k \geq 5 \), with edge lengths

\[
\begin{align*}
    c(a_i, a_{i+1}) &= c(d_i, d_{i+1}) := 1/k^2, & \text{for} & \ 1 \leq i \leq k, \\
    c(b_i, b_{i+1}) &= c(c_i, c_{i+1}) := 1, & \text{for} & \ i = 1, 2, 3, \\
    c(b_i, b_{i+1}) &= c(c_i, c_{i+1}) := 1/k^2, & \text{for} & \ 4 \leq i \leq k, \\
    c(a_i, b_i) &= c(c_i, d_i) := 1/k^2, & \text{for} & \ i = 1 \text{ or } 4 \leq i \leq k, \\
    c(a_i, b_i) &= c(c_i, d_i) := 1, & \text{for} & \ i = 2, 3, \\
    c(b_i, c_i) &= 1, & \text{for} & \ 1 \leq i \leq k,
\end{align*}
\]

and maximum possible length for all other edges such that the triangle inequality is not violated.

Figure 7.10 shows the basic structure of the graph. (Only some edges are shown.)

![Graph G_{4,k}](image)

**Figure 7.10:** The graph \( G_{4,k} \).

It can easily be seen that the graph \( G_{4,k} \) satisfies the triangle inequality using a similar reasoning as in the proof of Lemma 3.10.

**Lemma 7.12.** The Hamiltonian cycle

\[
H_{Opt} := (a_1, a_2, \ldots, a_k, b_k, b_{k-1}, \ldots, b_3, c_3, c_4, \ldots, c_k, d_k, d_{k-1}, \ldots, d_1, c_1, c_2, b_2, b_1, a_1)
\]

of length \( 6 + 4/k - 6/k^2 \) as shown in Figure 7.11 is optimal.
7.3. 4/3-Approximation of the $\Delta$-LM-TSP

Proof. If a Hamiltonian cycle for $G_{4,k}$ does not contain the vertices $b_2, b_3, c_2, c_3$ in consecutive (but arbitrary) order, then it always contains at least six edges of length at least 1. If a Hamiltonian cycle $H$ does contain them in consecutive order, let $v_1$ and $v_2$ be the vertex before and after the last of these four vertices in $H$. Independent of the position of $v_1$ and $v_2$, the path connecting them not containing the vertices $b_2, b_3, c_2, c_3$ always contains at least one other edge of length 1 (from some lower to some upper vertex).

Let $G'_{4,k}$ be the graph $G_{4,k}$ with the only change that the edge $\{b_2, c_2\}$ has length 3 instead of 1, i.e., $c(b_2, c_2) := 3$. Again, it can easily be seen that this graph satisfies the triangle inequality.

The optimal Hamiltonian cycle for $G'_{4,k}$,

$$H'_{\text{Opt}} := (a_1, a_2, \ldots, a_k, b_k, b_{k-1}, \ldots, b_4, c_4, c_5, \ldots, c_k, d_k, d_{k-1}, \ldots, d_1, c_1, c_2, c_3, b_3, b_2, b_1, a_1),$$
can be seen in Figure 7.12. Observe that it has the same length as $H_{\text{Opt}}$ has in $G_{4,k}$, i.e., $6 + 4/k - 6/k^2$.

We now show one possible implementation of Procedure 7.5 that, on inputs $G_{4,k}, G'_{4,k}, H_{\text{Opt}}$, returns a Hamiltonian cycle of length at least 8 in $G'_{4,k}$. Because $H_{\text{Opt}}$ has length $8 + 4/k - 6/k^2$ in $G'_{4,k}$, all we need to show is that the Christofides algorithm, when applied to $G'_{4,k}$, returns a Hamiltonian cycle of length at least 8.

The Christofides algorithm first computes the minimum spanning tree $T$ with the edges

$$\{\{a_i, a_{i+1}\}, \{d_i, d_{i+1}\} | 1 \leq i \leq k - 1\} \cup \{\{b_i, b_{i+1}\}, \{c_i, c_{i+1}\} | i = 1 \text{ or } 3 \leq i \leq k\} \cup \{\{a_1, b_1\}, \{a_k, b_k\}, \{b_3, c_3\}, \{c_1, d_1\}, \{c_k, d_k\}\},$$

as shown in Figure 7.13, resulting in the set of odd vertices $\{b_2, c_2\}$. Then, it computes the minimum matching $M = \{\{b_2, c_2\}\}$ for this set. By combining $T$ and $M$, it obtains the Hamiltonian cycle $H_{\text{Opt}}$ as shown in Figure 7.11 of length $8 + 4/k - 6/k^2$ in $G'_{4,k}$.

![Figure 7.13: A minimum spanning tree in $G'_{4,k}$. The odd vertices are circled.](image)

We have thus shown that, for arbitrarily small $\varepsilon > 0$, there exists an implementation $I$ of Procedure 7.5 such that

$$\frac{\text{cost}(I(G_{4,k}, G'_{4,k}, H_{\text{Opt}}))}{\text{Opt}_{\Delta=4}\cdot \text{TSP}(G_{4,k}, G'_{4,k}, H_{\text{Opt}})} \geq \frac{8 + 4/k - 6/k^2}{6 + 4/k - 6/k^2} \geq \frac{4}{3} - \varepsilon,$$

for sufficiently large $k$, i.e., we have established a lower bound of $4/3 - \varepsilon$ on the approximation ratio of Procedure 7.5 if $c_2(e) > c_1(e)$.

This and Lemma 7.12 immediately imply the optimality of $H'_{\text{Opt}}$ in $G'_{4,k}$.\footnote{This and Lemma 7.12 immediately imply the optimality of $H'_{\text{Opt}}$ in $G'_{4,k}$.}
Chapter 8

Conclusion

The goal of this thesis was to investigate the path matching Christofides algorithm for the traveling salesman problem and its modification for the Hamiltonian path problem with zero, one, and two fixed endpoints. We corrected an implementation error from [BHK+02] and showed that the upper bounds of $\frac{3}{2} \beta^2$ for the first three problems and $\frac{5}{2} \beta^2$ for the latter problem are tight by providing worst-case examples for all four algorithms. In particular, we could fill one gap in the following table.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Range</th>
<th>Upper bound</th>
<th>Lower bound</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cycle-cover</td>
<td>$\frac{1}{2} \leq \beta \leq \frac{2}{3}$</td>
<td>$\frac{\beta - 2}{3(\beta - 1)}$ [BHK+00]</td>
<td>$-\frac{1}{2}$</td>
</tr>
<tr>
<td>Christofides</td>
<td>$\frac{2}{3} \leq \beta \leq 1$</td>
<td>$\frac{3\beta^2}{3\beta^2 - 2\beta + 1}$ [BHK+00]</td>
<td>$\frac{2\beta^2 + 9 + \varepsilon}{6}$ [Spr08]</td>
</tr>
<tr>
<td>PMCA</td>
<td>$1 \leq \beta \leq 2$</td>
<td>$\frac{3}{2} \beta^2$ [BHK+02]</td>
<td>$-\frac{3}{2} \beta^2 - \varepsilon$</td>
</tr>
<tr>
<td>HCT$^3$ refined</td>
<td>$2 \leq \beta \leq 3$</td>
<td>$\beta^2 + \beta$ [And01]</td>
<td>$\beta^2 + \beta - \varepsilon$ [AB95]</td>
</tr>
<tr>
<td>Bender-Chekuri</td>
<td>$\beta \geq 3$</td>
<td>$4\beta$ [BC00]</td>
<td>$-\frac{1}{2}$</td>
</tr>
</tbody>
</table>

Table 8.1: The best currently known $\Delta_\beta$-TSP approximation algorithms.

There is, however, still a gap for the cycle-cover algorithm, the Christofides algorithm and the Bender-Chekuri algorithm. Maybe the worst-case examples
Chapter 8. Conclusion

presented in this thesis can also be used to obtain an improved lower bound on
the approximation ratio of the latter algorithm. More generally, the search for
better approximation algorithms for the TSP (and other hard approximation
problems) is a major open topic in research.

Furthermore, the \( \Delta \)-LM-TSP approximation algorithms devised by Böck-
enhauer et al. [BFH+07] and by Berg and Hempel [BH09] require further
analysis.


