Theoretical Computer Science 657 (2017) 111–126

Contents lists available at ScienceDirect

Theoretical Computer Science

www.elsevier.com/locate/tcs

Constant factor approximation algorithm for TSP satisfying a biased triangle inequality

Usha Mohan [a], Sivaramakrishnan Ramani [a], Sounaka Mishra [b,*]

[a] Department of Management Studies, Indian Institute of Technology Madras, Chennai, 600036, India
[b] Department of Mathematics, Indian Institute of Technology Madras, Chennai, 600036, India

ARTICLE INFO

Article history:
Received 20 August 2015
Received in revised form 18 September 2016
Accepted 21 September 2016
Available online 4 October 2016
Communicated by V.Th. Paschos

Keywords:
TSP
Relaxed triangle inequality
Approximation algorithms

ABSTRACT

In this paper, we study the approximability of a variant of the Traveling Salesman Problem called the `Biased-TSP`. In the `Biased-TSP`, the edge cost function violates the triangle inequality in a "controlled" manner. We give a $\frac{7}{2}$-factor approximation algorithm for this problem by a suitable modification of the double-tree heuristic.

© 2016 Elsevier B.V. All rights reserved.

1. Introduction

The Traveling Salesman problem (TSP) is one of the central problems in the field of combinatorial optimization. In the TSP we are given a complete undirected graph $G = (V, E)$ on $V = \{v_1, v_2, \ldots, v_n\}$ vertices and an edge-cost function $c : E \mapsto \mathbb{R}^+$. The aim is to find a Hamiltonian tour $\sigma = \langle v_{\pi(1)}, v_{\pi(2)}, \ldots, v_{\pi(n)}, v_{\pi(1)}\rangle$ with minimum cost $c(\sigma) = \sum_{i=1}^{n-1} c(v_{\pi(i)}, v_{\pi(i+1)}) + c(v_{\pi(n)}, v_{\pi(1)})$, where π is a permutation of $\{1, \ldots, n\}$. TSP has many practical applications and a list of them can be obtained from [3, Chapter 2]. One of these applications is drilling of a printed circuit board. In this drilling problem, the vertices correspond to the locations where a hole is to be drilled and the cost function is the time required to move the robotic arm between the holes. The minimum cost Hamiltonian tour corresponds to the sequence in which the holes are to be traversed so that the total time to traverse the sequence is minimum among all sequences.

TSP is a known NP-hard problem and admits no constant-factor approximation algorithm, assuming P $\neq$ NP [13]. However, TSP can be approximated within a constant factor when the cost function satisfies the triangle inequality (i.e., $c(u, w) \leqslant c(u, v) + c(v, w)$, for all $u, v, w \in V$) [8,4,11]. We shall denote TSP satisfying triangle inequality as $\triangle$-TSP.

Several attempts were made to design constant factor approximation algorithms for the TSP when the edge-cost function c violates the triangle inequality. Andreae and Bandelt [2] considered the approximability of TSP in which the edge-cost function satisfies a parametrized triangle inequality, denoted as $\triangle_\tau$-inequality, and defined as $c(u, w) \leqslant \tau(c(u, v) + c(v, w))$, for all $u, v, w \in V$ and $\tau \geqslant \frac{1}{2}$ is a fixed parameter. They specifically considered TSP instances satisfying the $\triangle_\tau$-inequality for $\tau \geqslant 1$ and called those TSP instances as the $\triangle_\tau$-TSP. Observe that in an instance of the $\triangle_\tau$-TSP with $\tau > 1$, triangle inequality might not be satisfied for any triples of vertices. Andreae and Bandelt [2] gave a modification of the double-tree

* Corresponding author.
E-mail addresses: ushamohan@iitm.ac.in (U. Mohan), ms12s016@smail.iitm.ac.in (S. Ramani), sounak@iitm.ac.in (S. Mishra).

http://dx.doi.org/10.1016/j.tcs.2016.09.017
0304-3975/© 2016 Elsevier B.V. All rights reserved.

algorithm (described as tree-algorithm in [12]) that approximates the $\triangle_\tau$-TSP with a factor of $\left(3\tau^2+\tau\right)/2$ for $\tau > 1$. They also observe that the approximation factor of Christofides' algorithm [8] increases more rapidly as τ increases.

In this paper, we study the approximability of the TSP when the edge costs satisfy the triangle inequality only for a specified subset of triples of vertices. We shall denote this version of the TSP as the `Biased-TSP`. Hence, the `Biased-TSP` is a special case of the $\triangle_\tau$-TSP and a generalization of $\triangle$-TSP.

1.1. Problem definition

Let $G = (V, E)$ be a complete undirected graph. Let $c : E \mapsto \mathbb{R}^+$ be the edge-cost function satisfying the triangle inequality. Let $\mathcal{P} = \{V_1, V_2\}$ be a 2-partition of the vertex set V and for $\beta : V_1 \times V_2 \mapsto [1, \infty)$, we define $\hat{c} : E \mapsto \mathbb{R}^+$ as follows:

$$\hat{c}(u,v) = \begin{cases} c(u,v) & \text{if } u, v \in V_1 \text{ or } u, v \in V_2 \\ \beta(u,v)c(u,v) & \text{if } u \in V_1, v \in V_2. \end{cases} \tag{1}$$

An edge cost function is said to satisfy the *biased*-triangle inequality with bias factor β if it satisfies the conditions as in (1).

An instance of the `Biased-TSP` is a triple $\mathcal{I} = (G, \mathcal{P}, \hat{c})$ and the goal is to compute a minimum cost Hamiltonian tour in the graph G with respect to the modified cost function $\hat{c}$. Here, we assume that there is no information about the bias factor function β and the cost function c satisfying the triangle inequality. But we have the information that each triple of vertices that is completely within the set V_1 or V_2 satisfies the triangle inequality, whereas each edge across the partition is scaled up by a factor of β which is unknown to us. It is the cost of the edges connecting the vertices across the partition for which the triangle inequality is potentially violated. In this paper, we study the approximability of the `Biased-TSP`.

The printed circuit board drilling problem mentioned in the previous section can be modelled as an instance of the `Biased-TSP`. Now, suppose we have two types of holes to be drilled. In this case the time taken to move from one type of hole to the other includes the time taken to physically move from one type to the other plus the time taken to change the tool at the location of the other type. Hence, in this case we can partition the vertices into two parts – one for each type of hole. The cost function satisfies the triangle inequality within each partition while it may potentially violate the triangle inequality for any triple of vertices across the partition.

1.2. Related work

Böckenhauer et al. [7] considered a variant of the TSP which is closely related to our work. They studied the approximability properties of the TSP in which the cost of exactly one edge is modified. In their work, they assumed that an instance (G, c) of the TSP (c satisfying the triangle inequality) is given. They considered the problem of finding an optimal tour for the modified instance (G, c') where c' is obtained from c by changing the cost of exactly one edge. They also assume that, they have the additional information of the optimal tour σ^* with respect to the original edge-cost function c. They proved that the problem is NP-complete. They formulated the problem as an instance of the $\triangle_\tau$-TSP and designed a $7\tau/(2+3\tau)$-approximation algorithm, where $\tau > 1$ is the parameter of $\triangle_\tau$-TSP. They further obtained a 1.4-factor approximation algorithm for the problem when the modified cost function c' also satisfies the triangle inequality. In both the cases, their algorithm depends on the knowledge of σ^*.

Shurbevski et al. [14] studied the approximability of TSP on bipartite graphs when the original cost function satisfying the quadrangle inequality is biased by a factor $\beta \geqslant 1$. Here, the biased factor β is the same for all the edges. They started with a complete bipartite graph and a cost function c satisfying the quadrangle inequality along with a bias factor of $\beta \geqslant 1$; i.e., when the Hamiltonian tour traverses from one partition V_1 to the other partition V_2, the edge-cost is scaled by a bias factor of β whereas there is no bias factor while traversing from V_2 to V_1. They extract a factor-2-approximation algorithm for the problem.

Another problem which is relevant in the context of the `Biased-TSP` is the $\triangle_\tau$-TSP. The current best approximation factor for the $\triangle_\tau$-TSP is 4τ given by Bender and Chekuri [5]. Better approximations are known for some smaller values of τ like, $(3/2)\tau^2$ for $1 < \tau \leqslant 2$ [6] and $\tau^2 + \tau$ for $2 < \tau \leqslant 3$ [1]. However, Bender and Chekuri [5] proved that $\triangle_\tau$-TSP cannot be approximated beyond $(1 + \epsilon\tau)$ for some $\epsilon > 0$, assuming P $\neq$ NP.

1.3. Our results

The `Biased-TSP` does not look very different from the $\triangle$-TSP and hence, one may be tempted to apply the approximation algorithms available for the $\triangle$-TSP to the `Biased-TSP` also. However, in Section 3, we construct instances of the `Biased-TSP` on which both the double-tree and Christofides' algorithm perform very poorly. Then, we modify the double-tree heuristic to obtain a 3.5-approximation algorithm for the `Biased-TSP`. The main technique used in achieving the constant factor approximation ratio is a short-cutting procedure that never shortcuts over an edge across the partition. Also, the set of edges across the partition present in the computed tour is a subset of the edges across the partition present in a minimum spanning tree of an instance of `Biased-TSP`. The algorithm and its analysis are presented in Section 4. Unlike the $\triangle_\tau$-TSP, where the approximation factor depends on the parameter τ, the approximation factor of the `Biased-TSP` is independent of β.

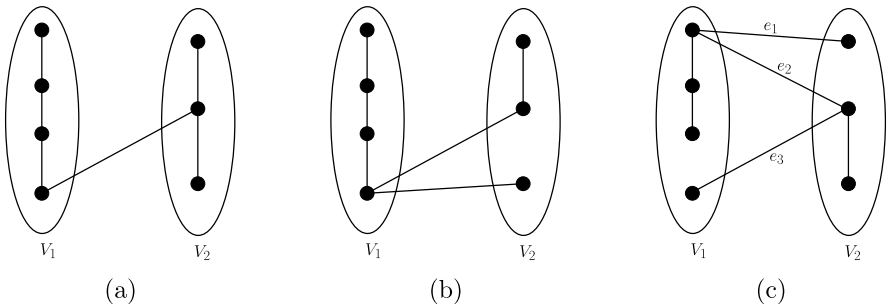


Fig. 1. Types of MST. (a) 0-MST with one cross-edge, (b) 0-MST with more than one cross-edge, (c) 1-MST. The edges e_1 and e_3 form a pair of independent cross-edges.

2. Notations and preliminaries

For notations related to graph theory we follow the notations given in [15]. For a graph G, we define $V(G)$ and $E(G)$ as the set of vertices and edges in G, respectively. However, we shall use V and E as the vertex set and the edge set of G, respectively, whenever there is no confusion. For a pair of graphs G and H we define $G \cup H$ as the new graph obtained by taking union of respective vertex sets and edge sets. Given a graph $G = (V, E)$ and a collection of edges $e_1, \ldots, e_k \in V \times V$ not in E, we define $G \cup \{e_1, \ldots, e_k\}$ as the graph $(V, E \cup \{e_1, \ldots, e_k\})$. Given graph $G = (V, E)$ and an edge weight function $c : E \to \mathbb{R}^+$, we define $c(G) = \sum_{e \in E} c(e)$.

Through out this paper, we consider $G = (V, E)$ to be a complete graph on n vertices. A 2-partition $\mathcal{P} = \{V_1, V_2\}$ is called a non-trivial 2-partition if both V_1 and V_2 have cardinality at least 2. An edge, $e \in E$, is called a cross-edge if its endpoints are in different partitions. A pair of edges e_1 and e_2 in E are called independent if the vertices they are incident to are all distinct. If the independent edges e_1 and e_2 are cross-edges with respect to $\mathcal{P}$ then we shall call this pair of edges independent cross-edges. A pair of independent cross-edges is called a pair of minimum cost independent cross-edges if the sum of their cost is minimum over all the pairs of independent cross-edges with respect to $\mathcal{P}$.

For a pair of vertices $u, v \in V$, a path $P(u, v)$ from u to v is a sequence of vertices such that every consecutive pair of vertices are adjacent in G. With respect to a tree $T = (V, E)$, we define a few notations which we shall use in this paper. Every pair of vertices u, v in a tree T has a unique path connecting u to v, and we shall denote this path as $P_T(u, v)$. For any two vertices u, v in T, we define the distance between u and v, $d(u, v)$, to be the number of edges in $P_T(u, v)$. For a vertex v in T, we define $depth(v) = \max_{u \in T} d(u, v)$ and $depth(T) = \min_{v \in T} depth(v)$. There can be at most two adjacent vertices u, v in T with their depths equal to $depth(T)$. We shall denote the *center* of T as the vertex v with $depth(v) = depth(T)$, provided there is an unique such vertex in T, otherwise we declare any one of such vertices as the center of T.

Given a graph $G = (V, E)$ with cost function $c : E \mapsto \mathbb{R}$, and $X \subset V$ the Steiner tree of G with respect to X is the minimum cost connected subgraph of G containing all the vertices of X. Computing the Steiner tree of a graph is NP-hard [9]. However, when G is a tree, the Steiner tree is unique and can be computed in polynomial time by deleting the leaf nodes not in X successively till all the leaf nodes are in X.

Let T be a rooted tree with r as its root and $v \neq r$ be any vertex in T. We define *ancestor* of v as the vertex u that is farthest from the root r with degree at least 3 in $P_T(r, v)$. We also define *level* of a vertex v of degree at least 3 in the rooted tree T as the number vertices of degree at least 3 in $P_T(r, v)$.

With respect to a non-trivial 2-partition $\mathcal{P} = \{V_1, V_2\}$ of V, we define two different types of spanning trees. A spanning tree T is called a 1-MST if it has at least one pair of independent cross-edges and is called a 0-MST if it has no pair of independent cross-edges. If T is a 0-MST then all the cross-edges of T are incident to a single vertex v. Depending on the number of such cross-edges, we further classify the type of 0-MSTs into two cases, one with exactly one cross-edge and other with more than one cross-edges. For an illustration we refer to Fig. 1.

Now, we state the following lemma which will be used in the analysis of the approximation ratio of our algorithm.

Lemma 1. *Consider a path $P(u, v)$ between two vertices u and v such that both are in V_1 or in V_2. Then $\hat{c}(u, v) \leqslant \hat{c}(P(u, v))$.*

Proof. Without loss of generality, we assume that both u, v in V_1. If $P(u, v)$ is completely contained in V_1, the result follows from triangle-inequality, since $\hat{c} = c$ within V_1, and c satisfies triangle-inequality.

Otherwise, since $\hat{c}(u, v) = c(u, v)$ and $c(u, v) \leqslant c(P(u, v))$, we have that $\hat{c}(u, v) \leqslant c(P(u, v))$. Since $c \leqslant \hat{c}$, the inequality holds. □

Throughout this paper, we denote the optimal Hamiltonian tour by σ^* and its cost by $\hat{c}(\sigma^*)$.

We make the following easy observation which we shall use later in this paper.

Observation 1. *Let $\mathcal{P}$ be a non-trivial 2-partition of V in a given instance $\mathcal{I}$ of the* `Biased-TSP`*. Let e_1 and e_2 be a pair of minimum cost independent cross-edges. Then, $\hat{c}(e_1) + \hat{c}(e_2) \leqslant \hat{c}(\sigma^*)$.*

The observation follows from the fact that in any Hamiltonian tour, there should be at least a pair of independent cross-edges, and the sum of cost of these edges cannot be smaller than the sum of cost of e_1 and e_2.

In the next two lemmas, we present approximation algorithms for constructing Hamiltonian path in a graph which will be used later in the paper. The first one is a folk result and the second one is Hoogeveen's algorithm [10]. However, the analysis of Lemma 3 is slightly different from the analysis presented in [10].

Lemma 2. *Let $T^* = (V, E(T^*))$ be a minimum spanning tree in $(G = (V, E), c)$ with edge cost function c satisfying the triangle inequality. For any pair of vertices $u, v \in V$, one can construct a Hamiltonian path $HP(u, v)$ from T^* such that $c(HP(u, v)) \leqslant 2c(T^*)$.*

Proof. We double all the edges in T^* except the edges in the unique path $P_T(u, v)$. The resulting multigraph contains an Eulerian path from u to v. From this path one can short-cut the edges to get Hamiltonian path $HP(u, v)$. Since the edge cost function satisfies the triangle inequality and each edge of T^* is used at most twice, we have that $c(HP(u, v)) \leqslant 2c(T^*)$. We shall refer to this algorithm as the *double-tree algorithm* in this paper. □

Next, we present Hoogeveen's algorithm [10] for constructing a Hamiltonian path. However, we bound the cost of the Hamiltonian path constructed by Hoogeveen's algorithm with respect to the optimal Hamiltonian tour in the graph rather than the optimal Hamiltonian path.

Lemma 3. [10] *For a graph $G = (V, E)$ with edge cost function c satisfying the triangle inequality, and any pair of vertices u, v, one can construct a Hamiltonian path $HP(u, v)$ such that $c(HP(u, v)) \leqslant 1.5\, c(\sigma^*)$ where σ^* is the minimum cost Hamiltonian tour in G.*

Proof. The Hamiltonian path $HP(u, v)$ is constructed using Hoogeveen's algorithm [10] which is as follows.

1. Compute a MST T in G. Let S be the set of odd degree internal vertices $(V \setminus \{u, v\})$ and even degree endpoints $(\{u, v\})$ in T.
2. Compute a minimum cost perfect matching M in $G[S]$ (the subgraph of G induced by the vertex set S) and add it to T to obtain an Eulerian path $EP(u, v)$ between u and v.
3. Short-cut $EP(u, v)$ to obtain the Hamiltonian path $HP(u, v)$.

Clearly, $c(HP(u, v)) \leqslant c(EP(u, v)) = c(T) + c(M) \leqslant c(\sigma^*) + c(M)$. The first inequality holds as c satisfies the triangle-inequality. Since $c(M) \leqslant 0.5\, c(\sigma^*)$, the result follows. □

We conclude this section with a simple lemma which we shall be using afterwards.

Lemma 4. *Let $\sigma^*_{V_1}$ and $\sigma^*_{V_2}$ be the minimum cost Hamiltonian tours in $G[V_1]$ and $G[V_2]$ with respect to the cost function $\hat{c}$, respectively. Then,*
(a) $\hat{c}(\sigma^*_{V_1}) \leqslant \hat{c}(\sigma^*)$, (b) $\hat{c}(\sigma^*_{V_2}) \leqslant \hat{c}(\sigma^*)$.

Proof. (a) In the optimal tour σ^*, we skip the vertices from V_2 to obtain a tour consisting of the vertices only from V_1 as follows. We start (and end) at a vertex $a_i \in V_1$ and traverse the vertices in the order given by σ^*; when we encounter any vertex from V_2, we skip it and go to the next vertex. We show that this short-cutting procedure does not increase the cost. Let $a_i, b_i, b_j, \ldots, b_p, a_j$ be a set of vertices appearing in the above order in σ^*, where $a_i, a_j \in V_1$, and $b_i, b_j, \ldots, b_p \in V_2$. Then, from Lemma 1, $\hat{c}(a_i, a_j) \leqslant \hat{c}(P)$ where P is the path $(a_i, b_i, b_j, \ldots, b_p, a_j)$.

Hence, the tour obtained on V_1 is of cost no greater than σ^*. Therefore, $\hat{c}(\sigma^*_{V_1}) \leqslant \hat{c}(\sigma^*)$.

The proof of part (b) is similar to that of (a). □

3. Bad instance for double-tree and Christofides' algorithm

In this section, we construct a family of instances of the `Biased-TSP` for which both the double-tree and Christofides' algorithm performs very badly. We start with a graph $G = (V, E)$ which is obtained by taking the metric closure of the simple unweighted path $P = (v_1, v_2, \ldots, v_n)$ on n vertices. It can be proved that the cost function $c(v_i, v_j) = j - i$ for $i < j$ satisfies the triangle inequality. Let $V_1 = \{v_1, v_2, \ldots, v_{n-1}\}$ and $V_2 = \{v_n\}$ be a 2-partition of the vertex set. For some fixed $\beta \geqslant 1$, define $\hat{c}$ as follows:

$$\hat{c}(v_i, v_j) = \begin{cases} c(v_i, v_j) & \text{for } 1 \leqslant i < j \leqslant n-1 \\ \beta c(v_i, v_n) & \text{for } 1 \leqslant i \leqslant n-1. \end{cases}$$

For odd n, $\sigma_1 = \langle v_1, v_3, \ldots, v_{n-2}, v_n, v_{n-1}, v_{n-3}, \ldots, v_4, v_2, v_1\rangle$ is a Hamiltonian tour with $\hat{c}(\sigma_1) = 2(n-3) + 3\beta + 1$. When n is even, $\sigma_2 = \langle v_1, v_3, \ldots, v_{n-1}, v_n, v_{n-2}, v_{n-4}, \ldots, v_4, v_2, v_1\rangle$ is a Hamiltonian tour with $\hat{c}(\sigma_2) = 2(n-3)+3\beta+1$. Hence, the cost of an optimal tour $\hat{c}(\sigma^*)$ is at most $2(n-3) + 3\beta + 1$. The minimum cost spanning tree in the graph G is the path $P = (v_1, v_2, \ldots, v_n)$. It can be observed that the double-tree and Christofides' algorithm outputs a tour $\sigma = \langle v_1, v_2, \ldots, v_n, v_1\rangle$ of cost $(n-2) + n\beta$. Hence, $\dfrac{\hat{c}(\sigma)}{\hat{c}(\sigma^*)} \geqslant \dfrac{(n-2)+n\beta}{2(n-3)+3\beta+1}$ which approaches at least $(1+\beta)/2$ as n goes to infinity.

4. Approximation algorithm for `Biased-TSP`

In this section, we show that the `Biased-TSP` can be approximated within a factor of 3.5. Our algorithm is similar to the double-tree algorithm. The main modification is the way in which the short-cutting procedure is implemented. From the definition of $\hat{c}$ in (1), it is clear that the triangle inequality is satisfied within V_1 and V_2 and hence, short-cutting across the edges with vertices completely contained in V_1 or V_2 will not increase the cost. But the cost may increase when the edges connecting the vertices across the partition are short-cut. This was exactly what happened for the instances considered in Section 3. We modify the short-cutting procedure so that we do not short-cut across the edges whose endpoints are in different partitions, and hence, we bound the approximation factor within a constant.

When one of the partitions V_1 or V_2 is a singleton, then we can obtain a 1.5-approximation algorithm for the `Biased-TSP`. Assume without loss of generality that V_1 is a singleton. In this case, the optimal solution will consist of a Hamiltonian path $HP^*_{V_2}$ on $G[V_2]$ and two cross-edges, e^*_1 and e^*_2. Our algorithm is as follows. First, we compute a MST T^* in $G[V_2]$. Next, we pick the two minimum cost cross-edges, e_1 and e_2. Let $x, y \in V_2$ be the vertices in V_2 on which e_1 and e_2 are incident to. Now, we compute a Hamiltonian path, $HP(x, y)$, in $G[V_2]$ between x and y using Lemma 3. Finally, we add e_1 and e_2 to $HP(x, y)$ to obtain a Hamiltonian cycle. The Hamiltonian path $HP(x, y)$, computed using Lemma 3, consists of adding a matching M in $G[V_2]$ to T^*. Hence, $\hat{c}(M) \leqslant 0.5\,\hat{c}(\sigma^*_{V_2}) \leqslant 0.5\,\hat{c}(\sigma^*)$ where the last inequality follows from Lemma 4. Also, $\hat{c}(T^*) + \hat{c}(e_1) + \hat{c}(e_2) \leqslant \hat{c}(HP^*_{V_2}) + \hat{c}(e^*_1) + \hat{c}(e^*_2) = \hat{c}(\sigma^*)$. Hence, the cost of the tour $HP(x, y) \cup \{(e_1, e_2)\}$ is at most 1.5 times the cost of the optimal solution.

In the rest of the paper we assume that the 2-partition $\mathcal{P}$ in $\mathcal{I} = (G, \mathcal{P}, \hat{c})$ is a non-trivial 2-partition. For such kind of instances, we prove that `Biased-TSP` can be approximated within a factor of 3.5. Our algorithm first finds a minimum spanning tree T^* in the given instance $(G, \mathcal{P}, \hat{c})$. Depending on whether it is a 0-*MST* or a 1-*MST* and depending on its depth, we consider several cases for designing our algorithm. For each case we design an algorithm that returns a Hamiltonian tour σ whose cost is at most 3.5 times the cost of the optimal tour σ^*.

4.1. Approximation algorithm for 0-MST

In this section, we design a 3-factor algorithm for the `Biased-TSP` when the minimum spanning tree T^* is a 0-*MST*. We further subdivide it into two cases: (1) 0-*MST* with only one cross-edge, and (2) 0-*MST* with more than one cross-edge. Each case is considered separately and a 3-approximation algorithm is obtained for them.

4.1.1. 0-MST with one cross-edge

Since T^* has only one cross-edge $e = (u, v) \in E(T^*)$, $T^* \setminus \{e\}$ has exactly two components T^*_1 and T^*_2 and they are minimum spanning trees of $G[V_1]$ and $G[V_2]$, respectively. The algorithm first computes a pair of minimum cost independent cross-edges $e_1 = (v^1_1, v^1_2)$ and $e_2 = (v^2_1, v^2_2)$. By using Lemma 2, we construct two Hamiltonian paths $HP_1(v^1_1, v^2_1)$ in $G[V_1]$ and $HP_2(v^1_2, v^2_2)$ in $G[V_2]$, from the spanning trees T^*_1 and T^*_2, respectively. Then we construct a Hamiltonian tour $\sigma = HP_1(v^1_1, v^2_1) \cup HP_2(v^1_2, v^2_2) \cup \{e_1, e_2\}$. A formal description of this algorithm is described in Algorithm 1.

Algorithm 1: Algorithm for `Biased-TSP` when T^* is a 0-*MST* with one cross-edge.

Input: 0-*MST* T^* with one cross-edge;
Output: A Hamiltonian tour σ;

Find a pair of minimum cost independent cross-edges $e_1 = (v^1_1, v^1_2)$ and $e_2 = (v^2_1, v^2_2)$ where $v^1_1, v^2_1 \in V_1$, and $v^1_2, v^2_2 \in V_2$;

Construct Hamiltonian paths $HP_1(v^1_1, v^2_1)$ and $HP_2(v^1_2, v^1_2)$ in $G[V_1]$ and $G[V_2]$ from $T^*[V_1]$ and $T^*[V_2]$, respectively;

Construct the tour $\sigma = HP_1(v^1_1, v^2_1) \cup HP_2(v^1_2, v^2_2) \cup \{e_1, e_2\}$;

return σ.

Theorem 1. *If the minimum spanning tree T^* has only one cross-edge then `Biased-TSP` can be approximated within a factor of 3.*

Proof. Let σ be the tour returned by Algorithm 1 and σ^* be the optimal tour. Let $\hat{c}(T^*[V_1])$ and $\hat{c}(T^*[V_2])$ denote the cost of $T^*[V_1]$ and $T^*[V_2]$ respectively. Let $HP_1(v^1_1, v^2_1)$ and $HP_2(v^1_2, v^2_2)$ be the Hamiltonian paths constructed from $T^*[V_1]$ and $T^*[V_2]$, respectively, as described in the proof of Lemma 2. Hence,

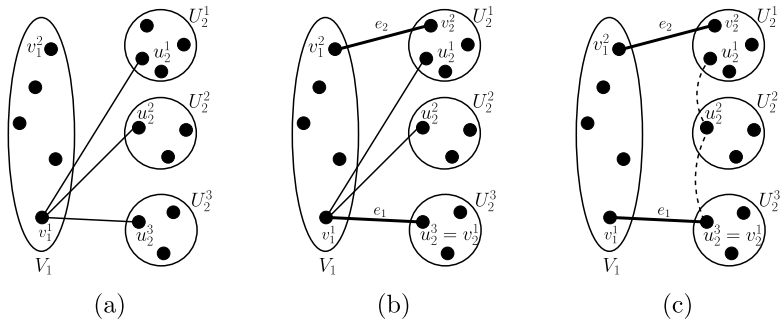


Fig. 2. An illustration of Algorithm 2. (a) 0-MST with more than one cross-edge, (b) 0-MST after adding the two minimum cost independent cross-edges, shown in bold, (c) the light dashed lines denote the edges used to merge the components in V_2.

$$\begin{aligned}\hat{c}(\sigma) &= \hat{c}(HP_1(v_1^1, v_1^2)) + \hat{c}(HP_2(v_2^1, v_2^2)) + \hat{c}(e_1) + \hat{c}(e_2)\\ &\leqslant 2\,\hat{c}(T^*[V_1]) + 2\,\hat{c}(T^*[V_2]) + \hat{c}(e_1) + \hat{c}(e_2)\\ &\leqslant 2\,\hat{c}(T^*) + \hat{c}(\sigma^*)\\ &\leqslant 3\,\hat{c}(\sigma^*).\end{aligned}$$

The inequality in the third line follows from Observation 1. □

4.1.2. 0-MST with more than one cross-edge

Now, we consider the case when the 0-MST has $p \geqslant 2$ cross-edges. Without loss of generality, let us assume that all these p cross-edges share the vertex $v_1^1 \in V_1$. Therefore, removal of the cross-edges from the MST T^* will result in one acyclic component $T^*[V_1]$, which is a spanning tree in $G[V_1]$, and p acyclic components $U_2^1, \ldots, U_2^p$ in $G[V_2]$. Let $u_2^i \in V(U_2^i)$, $1 \leqslant i \leqslant p$, be the vertices which are adjacent to v_1^1 in T^*. Then we compute a pair of minimum cost independent cross-edges $e_1 = (v_1^1, v_2^1)$ and $e_2 = (v_1^2, v_2^2)$. We observe that among these two edges, one of them will be incident to the common vertex $v_1^1 \in V_1$. If this was not true, then we can add one of e_1 and e_2 to T^* and remove an existing cross-edge from T^*, thereby decreasing the cost of T^*. This contradicts the fact that T^* is a MST.

From the spanning tree $T^*[V_1]$ and the pair of vertices v_1^1, v_1^2, by using Lemma 2, we construct a Hamiltonian path $HP_1(v_1^1, v_1^2)$ in $G[V_1]$. Next, we construct a Hamiltonian path $HP_2(v_2^1, v_2^2)$ in $G[V_2]$. In order to do so, we need a spanning tree in $G[V_2]$ and we construct such a spanning tree by adding $p-1$ edges to the components $U_2^1, \ldots, U_2^p$. If v_2^1 and v_2^2 are in two different components, then order the components $U_2^1, \ldots, U_2^p$ such that v_2^1 and v_2^2 are in the first and last component, respectively. Then we introduce the edges (u_2^i, u_2^{i+1}), for $1 \leqslant i \leqslant p-1$ to get a spanning tree of $G[V_2]$. In the other case, if both the vertices v_2^1, v_2^2 are in a single component, say U_2^k, then we introduce the edges (u_2^i, u_2^{i+1}), for $1 \leqslant i \leqslant p-1$ to get a spanning tree of $G[V_2]$. We denote this spanning tree as C. From C and the pair of vertices v_2^1, v_2^2, we construct a Hamiltonian path $HP_2(v_2^1, v_2^2)$ as described in the proof of Lemma 2. For an illustration, we refer to Fig. 2 and a formal description of this algorithm is presented in Algorithm 2.

Algorithm 2: Algorithm for `Biased-TSP` when T^* is 0-MST with more than one cross-edge.

Input: 0-MST T^* with more than one cross-edge;
Output: A Hamiltonian tour;

Remove the cross-edges from T^* to obtain the components $\{T^*[V_1], U_2^1, \ldots, U_2^p\}$ where $V(U_2^i) \subset V_2$. Let $u_2^k \in V(U_2^k)$ for all $1 \leqslant k \leqslant p$ be the vertex on which the cross-edge is incident;
Find a pair of minimum cost independent cross-edges $e_1 = (v_1^1, v_2^1)$ and $e_2 = (v_1^2, v_2^2)$;
Using Lemma 2 on $T^*[V_1]$ and the pair of vertices v_1^1, v_1^2, construct a Hamiltonian path $HP_1(v_1^1, v_1^2)$ in $G[V_1]$;
Construct a spanning tree C of $G[V_2]$ by adding edges (u_2^i, u_2^{i+1}), for $1 \leqslant i \leqslant p-1$, to $T^*[V_2]$, depending on whether v_2^1 and v_2^2 are in one component or not;
Using Lemma 2 on C and the pair of vertices v_2^1, v_2^2, construct a Hamiltonian path $HP_2(v_2^1, v_2^2)$ in $G[V_2]$;
Construct the tour $\sigma = HP_1(v_1^1, v_1^2) \cup HP_2(v_2^1, v_2^2) \cup \{e_1, e_2\}$;
return σ.

Theorem 2. *If the minimum spanning tree T^* is a 0-MST with more than one cross-edge then* `Biased-TSP` *can be approximated within a factor of 3.*

Proof. Let $v_1^1 \in V_1$ be the vertex upon which all the cross-edges are incident to. For any two components U_2^i and U_2^j, let u_2^i and u_2^j be the vertices on which the cross-edges (v_1^1, u_2^i) and (v_1^1, u_2^j) are incident to. Since u_2^i and u_2^j are two distinct vertices in V_2 and $P_{T^*}(u_2^i, u_2^j) = (u_2^i, v_1^1, u_2^j)$ is the path between u_2^i and u_2^j, by Lemma 1, we have that $\hat{c}(u_2^i, u_2^j) \leqslant \hat{c}(P_{T^*}(u_2^i, u_2^j)) = \hat{c}(u_2^i, v_1^1) + \hat{c}(v_1^1, u_2^j)$. Therefore,

$$\begin{aligned}\sum_{i=1}^{p-1} \hat{c}(u_2^i, u_2^{i+1}) &\leqslant \sum_{i=1}^{p-1} \left(\hat{c}(u_2^i, v_1^1) + \hat{c}(v_1^1, u_2^{i+1})\right) \\ &\leqslant 2\,\hat{c}(T_c^*) - \hat{c}(u_2^1, v_1^1) - \hat{c}(v_1^1, u_2^p) \\ &\leqslant 2\hat{c}(T_c^*), \end{aligned} \tag{2}$$

where T_c^* is the set of cross-edges of T^*.

Algorithm 2 uses the double-tree algorithm to construct the Hamiltonian path $HP_2(v_2^1, v_2^2)$ in the spanning tree, C, obtained by merging the components from $T^*[V_2]$. Hence, the edges $\cup_{i=1}^{p-1} \{(u_2^i, u_2^{i+1})\}$ will be used only once. Therefore,

$$\begin{aligned}\hat{c}(HP_2(v_2^1, v_2^2)) &\leqslant 2\sum_{i=1}^{p} \hat{c}(U_2^i) + \sum_{i=1}^{p-1} \hat{c}(u_2^i, u_2^{i+1}) \\ &\leqslant 2\sum_{i=1}^{p} \hat{c}(U_2^i) + 2\,\hat{c}(T_c^*),\end{aligned}$$

where the last inequality follows from (2).

Hence, if $HP_1(v_1^1, v_1^2)$ is the Hamiltonian path constructed by the algorithm on $T^*[V_1]$, and e_1 and e_2 are the pair of minimum cost independent cross-edges, then the cost of the tour σ is given as

$$\begin{aligned}\hat{c}(\sigma) &= \hat{c}(HP_1(v_1^1, v_1^2)) + \hat{c}(HP_2(v_2^1, v_2^2)) + \hat{c}(e_1) + \hat{c}(e_2) \\ &\leqslant 2\,\hat{c}(T^*[V_1]) + 2\sum_{i=1}^{p} \hat{c}(U_2^i) + 2\,\hat{c}(T_c^*) + \hat{c}(e_1) + \hat{c}(e_2) \\ &\leqslant 2\,\hat{c}(T^*) + \hat{c}(\sigma^*) \\ &\leqslant 3\,\hat{c}(\sigma^*),\end{aligned}$$

where the penultimate inequality follows from Observation 1. □

From Theorem 1 and Theorem 2, we get the following.

Theorem 3. *If an instance of the* `Biased-TSP` *has a 0-MST, then the* `Biased-TSP` *can be approximated within a factor of 3.*

4.2. Approximation algorithm for 1-MST

Now, we focus on the case when the MST T^* is a 1-MST. Here, the removal of the cross-edges from T^* would result in acyclic components $C_1 = \{U_1^1, \ldots, U_1^l\}$ where for all $1 \leqslant i \leqslant l$, $V(U_1^i) \subseteq V_1$, and $C_2 = \{U_2^1, \ldots, U_2^p\}$ where for all $1 \leqslant i \leqslant p$, $V(U_2^i) \subseteq V_2$.

Definition 1 *(Component graph). From the MST T^*, and the set of cross-edges T_c^* of T^*, we construct a graph $H = (V(H), E(H))$ defined as the component graph as follows. We treat each component of $(V, E(T^*) \setminus T_c^*)$ as a single vertex and define $V(H) = \{U_1^i \mid 1 \leqslant i \leqslant l\} \cup \{U_2^j \mid 1 \leqslant j \leqslant p\}$, and $E(H) = \{(U_1^i, U_2^j) \mid \exists$ an edge $(u, v) \in T_c^*$ such that $u \in U_1^i$ and $v \in U_2^j\}$.*

An illustration of the component graph is presented in Fig. 3. It is important to observe that the component graph H of T^* with respect to a set of edges S of T^* is a unique tree. For any two components A and B in $(V, E(T^*) \setminus S)$ which are adjacent in H, there is a unique edge in S whose end points are in components A and B. Hence, a cycle in H implies a cycle in T^*, thereby contradicting the acyclicity of T^*.

Given an input $(G, \hat{c})$ of `Biased-TSP`, we first compute a minimum spanning tree T^*. We assume that T^* is a 1-*MST*. Then, we compute the component graph H of T^* with respect to the cross-edges of T^*. Depending on the *depth*(H), we consider three cases for constructing a Hamiltonian tour. These three cases depend on the values of *depth*(H) as 1, 2 or more than 2.

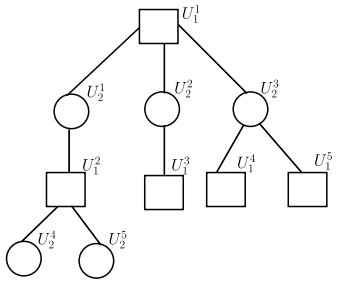


Fig. 3. An illustration of a component graph. The circular vertices denote the components from one partition, and the square vertices denote the component from the other partition.

4.2.1. Algorithm when depth of H is one

Since the $depth(H) = 1$, one among $T^*[V_1]$ or $T^*[V_2]$ should be a single component. Without loss of generality, we assume that $T^*[V_1]$ is a single component. Let $\{U_2^1, \ldots, U_2^p\}$ be the components of $T^*[V_2]$. However, unlike the 0-MST with more than one cross-edge, the components $U_2^1, \ldots U_2^p$ are not all incident to the same vertex in V_1. Hence, let $\{v_1^1, \ldots, v_1^l\} \subseteq V_1$, and $u_2^i \in V(U_2^i)$ for all $1 \leqslant i \leqslant p$ be the vertices from V_1 and V_2 on which the cross-edges are incident. We denote the vertices on which cross-edges are incident as marked vertices and we shall refer to $M = \{v_1^1, \ldots, v_1^l\}$ as the set of marked vertices in V_1.

First, we describe a procedure called `Path-Merge`$(P_{T^*}(v_1^i, v_1^j))$ which we shall use frequently in our algorithms. Here, $P_{T^*}(v_1^i, v_1^j)$ is the path in $T^*[V_1]$ between the marked vertices v_1^i and v_1^j. This procedure merges the components of $T^*[V_2]$ connected to the marked vertices in $P_{T^*}(v_1^i, v_1^j)$ and then attaches the merged component to $P_{T^*}(v_1^i, v_1^j)$.

Procedure: `Path-Merge`$(P_{T^*}(v_1^i, v_1^j))$.

Identify all the marked vertices $v_1^i, v_1^{i+1}, \ldots, v_1^j$ in $P_{T^*}(v_1^i, v_1^j)$. Note that $P_{T^*}(v_1^i, v_1^j)$ may contain some unmarked vertices too;

Let $U_2^1, U_2^2, \ldots, U_2^k$ be the ordered components in $T^*[V_2]$ as they are connected to the marked vertices in $P_{T^*}(v_1^i, v_1^j)$. If there are more than one component connected to a marked vertex, then arrange the components corresponding to that vertex in an arbitrary order;

For all $1 \leqslant i \leqslant k-1$, merge the components U_2^i, U_2^{i+1} by adding the edge between the vertices u_2^i and u_2^{i+1} to obtain a single component $C = \bigcup_{i=1}^k U_2^i \cup \{(u_2^i, u_2^{i+1}) : 1 \leqslant i \leqslant k-1\}$;

Using the double-tree algorithm of Lemma 2, construct a Hamiltonian path $HP_2(u_2^1, u_2^k)$ in $G[V(C)]$;

return $P_{T^*}(v_1^i, v_1^j) \cup HP_2(u_2^1, u_2^k) \cup \{(v_1^i, u_2^1), (v_1^j, u_2^k)\}$.

Lemma 5. *Let G' be the graph obtained by the procedure* `Path-Merge`$(P_{T^*}(v_1^i, v_1^j))$. *Let T_c^* denotes the cross-edges incident to the marked vertices in $P_{T^*}(v_1^i, v_1^j)$. Then,*

(i) G' is a Hamiltonian cycle on $P_{T^*}(v_1^i, v_1^j) \cup U_2^1 \cup \cdots \cup U_2^k$.
(ii) The degree of v_1^i and v_1^j in G' is one more than their respective degrees in $P_{T^*}(v_1^i, v_1^j)$.
(iii) $\hat{c}(G') \leqslant 2\left(\hat{c}(P_{T^*}(v_1^i, v_1^j)) + \sum_{i=1}^k \hat{c}(U_2^i) + \hat{c}(T_c^*)\right)$.

Proof. (i) The proof of this part is straightforward.
(ii) This follows from (i) and the fact that v_1^i and v_1^j are the endpoints of $P_{T^*}(v_1^i, v_1^j)$.
(iii) First we bound the cost of the edges used to merge the components $U_2^1, \ldots, U_2^k$. Let (u_2^i, u_2^{i+1}) be the edge used to merge the components U_2^i and U_2^{i+1}. Let $x, y \in M$ be the neighbours of u_2^i and u_2^{i+1} in $P_{T^*}(v_1^i, v_1^j)$ respectively. By Lemma 1, $\hat{c}(u_2^i, u_2^{i+1}) \leqslant \hat{c}(P_{T^*}(x, y)) + \hat{c}(x, u_2^i) + \hat{c}(y, u_2^{i+1})$ where $\hat{c}(P_T^*(x, y))$ is the cost of the path $P_{T^*}(x, y)$. Therefore,

$$\sum_{i=1}^{k-1} \hat{c}(u_2^i, u_2^{i+1}) \leqslant \hat{c}(P_{T^*}(v_1^i, v_1^j)) + 2\,\hat{c}(T_c^*) - \hat{c}(v_1^i, u_2^1) - \hat{c}(v_1^j, u_2^k).$$

Since the Hamiltonian path $HP_2(u_2^1, u_2^k)$ is constructed using the double-tree algorithm, the edges $\cup_{i=1}^{k-1}\{(u_2^i, u_2^{i+1})\}$ will be used only once. Hence,

$$\hat{c}(HP_2(u_2^1, u_2^k)) \leqslant 2\sum_{i=1}^{k}\hat{c}(U_2^i) + \sum_{i=1}^{k-1}\hat{c}(u_2^i, u_2^{i+1})$$
$$\leqslant 2\sum_{i=1}^{k}\hat{c}(U_2^i) + \hat{c}(P_{T^*}(v_1^i, v_1^j)) + 2\,\hat{c}(T_c^*) - \hat{c}(v_1^i, u_2^1) - \hat{c}(v_1^j, u_2^k).$$

Since $G' = P_{T^*}(v_1^i, v_1^j) \cup HP_2(u_2^1, u_2^k) \cup \{(v_1^i, u_2^1), (v_1^j, u_2^k)\}$,

$$\hat{c}(G') = \hat{c}(P_{T^*}(v_1^i, v_1^j)) + \hat{c}(HP_2(u_2^1, u_2^k)) + \hat{c}(v_1^i, u_2^1) + \hat{c}(v_1^j, u_2^k)$$
$$\leqslant 2\left(\hat{c}(P_{T^*}(v_1^i, v_1^j)) + \sum_{i=1}^{k}\hat{c}(U_2^i) + \hat{c}(T_c^*)\right). \quad \square$$

Now we focus on the case when the vertices $\{v_1^1, \ldots, v_1^l\}$ lie on a path in $T^*[V_1]$. We prove that a 2-factor algorithm can be obtained in this case.

Lemma 6. *If all the marked vertices $\{v_1^1, \ldots, v_1^l\}$ lie on a path in $T^*[V_1]$, then the* `Biased-TSP` *can be approximated within a factor of 2.*

Proof. Let $P_{T^*}(v_1^1, v_1^l)$ be the path in T^* on which the marked vertices $\{v_1^1, \ldots, v_1^l\}$ lie. Let $U_2^1, \ldots, U_2^p$ be the components from $T^*[V_2]$ and T_c^* the set of cross-edges in T^*. The algorithm can now be described as follows.

1. Apply the procedure `Path-Merge`$(P_{T^*}(v_1^1, v_1^l))$ to obtain a Hamiltonian cycle H on $P_{T^*}(v_1^1, v_1^l) \cup \bigcup_{i=1}^{p} U_2^i$.
2. Double all the edges of $T^*[V_1] \setminus P_{T^*}(v_1^1, v_1^l)$ and add them to H to obtain an Eulerian graph H'.
3. Construct an Eulerian tour in H'. Starting from a cross-edge of H', short-cut the Eulerian tour to obtain a Hamiltonian tour σ in G.

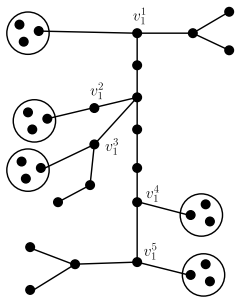
It is easy to see that the graph H' is Eulerian. For the approximation factor, we first bound the cost of the Hamiltonian cycle H obtained using `Path-Merge`$(P_{T^*}(v_1^i, v_1^j))$. From Lemma 5, we have $\hat{c}(H) \leqslant 2[\hat{c}(P_{T^*}(v_1^1, v_1^l)) + \sum_{i=1}^{p}\hat{c}(U_2^i) + \hat{c}(T_c^*)]$. Since $\hat{c}(H') = 2[\hat{c}(T^*[V_1]) - \hat{c}(P_{T^*}(v_1^1, v_1^l))] + \hat{c}(H)$, the cost of the Eulerian tour in H' is at most $2[\hat{c}(T^*[V_1]) + \sum_{i=1}^{p}\hat{c}(U_2^i) + \hat{c}(T_c^*)] \leqslant 2\,\hat{c}(T^*) \leqslant 2\,\hat{c}(\sigma^*)$. Since all the short-cuts are done in V_1, the cost of the Hamiltonian tour σ is at most the cost of the Eulerian tour. $\square$

Next, we consider the case when the vertices $\{v_1^1, \ldots, v_1^l\}$ do not lie in a path in $T^*[V_1]$. Here, we cannot merge the components $U_2^1, \ldots, U_2^p$ arbitrarily, since in an arbitrary merging, the edges on the path connecting the components may be used more than twice.

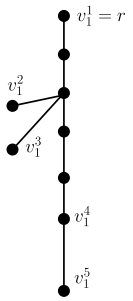
To avoid the edges in $T^*[V_1]$ being used more than twice, we attach the components of $T^*[V_2]$ to $T^*[V_1]$ in a systematic manner from leaf nodes towards the center of a connected subgraph of $T^*[V_1]$. In order to do this, we first construct a Steiner tree T of $T^*[V_1]$ with respect to the marked vertices $\{v_1^1, \ldots, v_1^l\}$.

Let H' be the graph obtained by doubling all the edges in the graph $(V_1, E(T^*[V_1]) \setminus E(T))$. We attach the components in $T^*[V_2]$ one by one to H' by traversing the tree T in a systematic manner. We pick a leaf node r in T and declare it as its root. Then we find a vertex v of degree at least 3 which is at a maximum distance from the root. We break ties arbitrarily. We consider the subtree T_v rooted at v and the paths hanging at the children of v. For every branch hanging from a child of v with more than one marked vertex, we add to H' the components of $T^*[V_2]$ connected by cross-edges to the marked vertices in that branch by using the procedure `Path-Merge`$(P_T(v_1^1, v_1^k))$ where $P_T(v_1^1, v_1^k)$ is the path containing all the marked vertices in that branch. Finally, we add $P_T(v_1^1, v) \cup \{(v_1^1, v)\}$ to H'. For the branches hanging at v with exactly one marked vertex, we pair them by excluding at most one branch. Let x_1 and y_1 be the marked vertices in a pair of branches. By using the procedure `Path-Merge`$(P_T(x_1, y_1))$ we add to H' the components of $T^*[V_2]$ connected by cross-edges to the marked vertices x_1 and y_1. Finally, we remove all the branches of T_v from T. However, if there is a branch with exactly one marked vertex and not paired with any other branches, we keep it in T. We call this procedure of attaching the components of $T^*[V_2]$ which are connected to the marked vertices of a subtree T_v to H' as `Tree-Merge`(T_v).

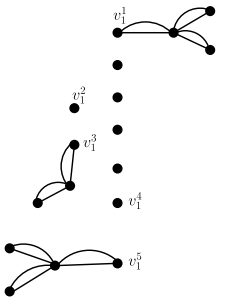
We repeat this procedure unless $V(T) = \emptyset$ or $V(T) = \{r\}$, where r is the root of the tree. If $V(T) = \emptyset$, then all the components from $T^*[V_2]$ have been attached to H'. In this case, we find an Eulerian tour in H' and short-cut it appropriately to get a Hamiltonian tour σ in G such that the cross-edges in H' and σ are same. When $V(T) = \{r\}$, the component(s) of $T^*[V_2]$ connected to r are not added to H'. In this case, we find the marked vertex x closest to r and add to H' the component(s) of $T^*[V_2]$ connected to r by using the procedure `Path-Merge`$(P_T(r, x))$. Here the procedure `Path-Merge`$(P_T(r, x))$ needs at least one component of $T^*[V_2]$ connected to x by cross-edges. Since all such components are already taken care, we choose $y \in V_2$, the only neighbour of x in H', as a component in V_2 adjacent to x. Note that in `Path-Merge`$(P_T(r, x))$, the



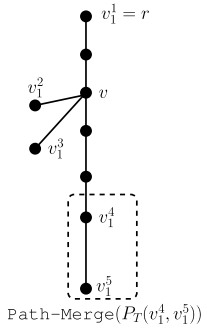
(a) Minimum spanning tree T^*. The big circles represent components of $T^*[V_2]$.



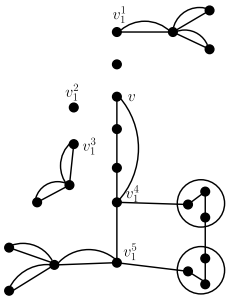
(b) Steiner tree T with v_1^1 as root and $v_1^1, \ldots, v_1^5$ as its marked vertices.



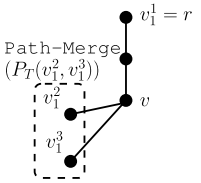
(c) Initial H' graph.



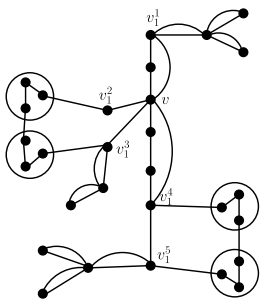
(d) v is the only ancestor. `Path-Merge`$(P_T(v_1^4, v_1^5))$ will be applied on T.



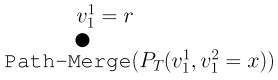
(e) H' after application of `Path-Merge`$(P_T(v_1^4, v_1^5))$.



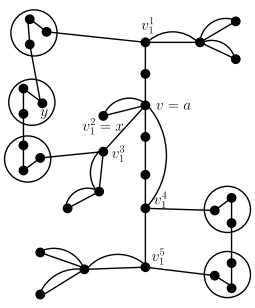
(f) Steiner tree T after the application of `Path-Merge`$(P_T(v_1^4, v_1^5))$. `Path-Merge`$(P_T(v_1^2, v_1^3))$ will be applied next.



(g) H' after application of `Path-Merge`$(P_T(v_1^2, v_1^3))$.



(h) Steiner tree T after the application of `Path-Merge`$(P_T(v_1^2, v_1^3))$. `Path-Merge`$(P_T(v_1^1, v_1^2))$ will be applied next.



(i) Final H'.

Fig. 4. A sketch of the construction of the Eulerian graph H' in Algorithm 3.

path $P_T(r, x)$ will not be added to H'. Then, we remove from H' the cross-edge (x, y) and add to H' a copy of the edge $e = (x, a)$ where a is the first vertex from r with degree at least three in the original Steiner tree T. Then, we construct an Eulerian tour in H' and short-cut to obtain a Hamiltonian tour in G. The algorithm is formally described in Algorithm 3. An illustration of this procedure is presented in Fig. 4.

Lemma 7. *The graph H' constructed by Algorithm 3 is Eulerian.*

Algorithm 3: Algorithm for computing a Hamiltonian tour for a MST with $depth = 1$ where the marked vertices $M = \{v_1^1, \ldots v_1^l\}$ do not lie on a path.

Input: An input $(G, \mathcal{P}, \hat{c})$ of `Biased-TSP`
Output: A Hamiltonian tour σ

Compute a MST T^* of G;
Compute the Steiner subtree T of $T^*[V_1]$ with all its leaf nodes in M;
Root T at a leaf vertex r of T and T' be a copy of T;
Let H' be the graph obtained by doubling all the edges of the graph $(V_1, E(T^*[V_1]) \setminus E(T))$;
while $V(T) \neq \emptyset$ *and* $V(T) \neq \{r\}$ **do**
  For all the leaf nodes of T, compute their ancestors and the level of their ancestors;
  if *all the leaves have no ancestor (T is a path)* **then**
    $H' = H' \cup$ `Path-Merge`$(P_T(v_1^1, v_1^k))$ where $P_T(v_1^1, v_1^k)$ is the path in T on which all its marked vertices lie;
    Delete T;
  end
  else
    Find the ancestor v with the maximum level (break ties arbitrarily);
    Run the procedure `Tree-Merge`(T_v) on the subtree T_v rooted at v;
  end
  if v *is unmarked and all the branches of* T_v *are removed* **then**
    Let u be the first vertex in the path $P_T(v, r)$ which is either marked or an ancestor of v;
    $H' = H' \cup P_T(u, v) \cup \{(v, u)\}$;
    $T = T \setminus P_T(v, u_1)$ where u_1 is the child of u in $P_T(v, u)$;
  end
end
if $V(T) = \{r\}$ **then**
  Let a be the first vertex from r with degree at least three in T'. Let x be the marked vertex nearest to r in T';
  $H' = \big(H' \cup$ `Path-Merge`$(P_T(r, x))\big) \setminus \{(a, r)\}$;
  From H' remove the cross-edge incident to x;
  Add a copy of (x, a) to H';
end
Construct an Eulerian tour σ_e in H';
Short-cut σ_e to obtain a Hamiltonian tour σ such that both have the same set of cross-edges;
return σ.

Proof. Before the start of the while loop of Algorithm 3, H' is the graph obtained by doubling all the edges of the graph $(V_1, E(T^*[V_1]) \setminus E(T))$. Hence, degree of each vertex in H' is even and H' is disconnected. At every iteration of the while loop, we delete a path from Steiner tree T and add a related graph (with every vertex of even degree) to H'. Therefore, it is enough to prove that the final graph H' is connected and spans all the vertices of $V_1 \cup V_2$.

Let T_v be the subtree of T whose ancestor v is at the maximum level. Let $P = (v, \ldots, v_1^1, \ldots, v_1^k)$ be a branch of T_v with more than one marked vertex. Let $P_{T_v}(v_1^1, v_1^k)$ be the path in P containing all the marked vertices of P. Note that v_1^k is a leaf node of T. Initially, the degree of all the vertices of $P_{T_v}(v_1^1, v_1^k)$ except v_1^1 and v_1^k are even. After Algorithm 3 adds to H' the output of the procedure `Path-Merge`$(P_{T_v}(v_1^1, v_1^k))$, we note from Lemma 5, the degree of v_1^1 and v_1^k becomes even in H'. Hence, after adding $P_{T_v}(v, v_1^1)$ and (v_1^1, v) to H', the degree of all the vertices of P are even in H'. Note that `Path-Merge`$(P_{T_v}(v_1^1, v_1^k))$ also adds to H' all the vertices in the connected components of $T^*[V_2]$ which are connected to the marked vertices in $P_{T_v}(v_1^1, v_1^k)$. From Lemma 5, the degree of all these vertices are also even. Among the branches of T_v containing exactly one marked vertex, Algorithm 3 pairs them by excluding at most one branch to obtain a set of edge disjoint paths DP_v. Therefore, after running the procedure `Path-Merge` on all the paths $P \in DP_v$, the degree of all the vertices of $P \in DP_v$ in H' is even.

Since this procedure is repeated till $V(T) = \emptyset$ or $V(T) = \{r\}$ and every iteration of the while loop keeps adding a graph of even degree to H', the degree of each vertex of the graph H' obtained at the termination of the while loop is even. All the edges of the Steiner tree T are also added to H' and hence, H' is a connected graph of even degree at the termination of the while loop. If $V(T) = \emptyset$, then all the components of $T^*[V_2]$ have been attached to H'. Hence, H' spans $V_1 \cup V_2$. Now, we consider the case when $V(T) = \{r\}$. Let a be the first vertex from r with degree at least three and x be the marked vertex nearest to r in the original Steiner tree T. In this case, Algorithm 3 adds the output of `Path-Merge`$(P_T(r, x))$ (here the `Path-Merge` is slightly different and it is described just before Algorithm 3) to H'. Finally, we remove the cross-edges incident to x and the edge (a, r) from H' and add a copy of the edge (x, a) to H'. Hence, all the components from $T^*[V_2]$ are added to H' and degree of each vertex remains even. □

Lemma 8. *Algorithm 3 is a 3-approximation algorithm for the instances of* `Biased-TSP` *whose minimum spanning tree is a 1-MST and the depth of its component graph is 1.*

Proof. We prove this by showing that the total cost of the edges in the graph H', constructed by Algorithm 3, is at most three times the cost of the optimal Hamiltonian tour. By Lemma 7, H' is Eulerian and hence, the cost of the Eulerian tour in

H' will also be at most three times the cost of the optimal solution. Since the Hamiltonian tour is obtained by a short-cut of the edges completely contained in the partition V_1, the claim of the lemma follows.

Before the beginning of the while loop of Algorithm 3, we have that $\hat{c}(H') = 2(\hat{c}(T^*[V_1]) - \hat{c}(T))$. Since H' is modified in every iteration of the while loop, we bound the total increase in cost of H' by bounding the increase in cost of H' in every iteration of the while loop.

In a particular iteration of the while loop, let T_v be a subtree of T whose ancestor v is at the maximum level. For any branch P of T with more than one marked vertex, an edge is added between the first marked vertex of P and v. From the triangle inequality, the cost of this edge is at most the cost of the edges between v and the first marked vertex of P. Apart from adding these edges, the procedure `Path-Merge` is applied on P and all the branches of T_v which are paired. Hence, from Lemma 5, the increase in cost of H' after this iteration is at most twice the cost of T_v plus twice the cost of the components of $T^*[V_2]$ connected to T_v plus twice the cost of the cross-edges incident to the vertices of T_v.

The while loop terminates when $V(T) = \emptyset$ or $V(T) = \{r\}$. If $V(T) = \emptyset$, then all the components of $T^*[V_2]$ has been attached to H'. Hence,

$$\hat{c}(H') \leqslant 2\left(\hat{c}(T^*[V_1]) - \hat{c}(T)\right) + 2\left(\hat{c}(T) + \sum_{i=1}^{p} \hat{c}(U_2^i) + T_c^*\right)$$
$$\leqslant 2\left(\hat{c}(T^*[V_1]) + \sum_{i=1}^{p} \hat{c}(U_2^i) + T_c^*\right) = 2\hat{c}(T^*) \leqslant 2\hat{c}(\sigma^*).$$

When $V(T) = \{r\}$, let x be the nearest marked vertex from r and a be the first vertex from r with degree at least three. In this case, we apply `Path-Merge`$(P_T(r, x))$ (here the `Path-Merge` is slightly different and it is described just before Algorithm 3), remove the cross-edge incident to x, and add the edge (x, a). The procedure `Path-Merge`$(P_T(r, x))$ can be viewed in an equivalent way as follows. Let $C_r \subset T^*[V_2]$ denotes the set of component(s) connected to r by the set of cross-edge(s) $T^*_{C_r}$. First, we apply `Path-Merge`$(P_T(r, r))$ to obtain a Hamiltonian path on the component(s) from C_r along with a cross-edge incident to r. In the next step, we identify the vertex $y \in V_2$ the neighbour of x in H' and merge the Hamiltonian path to y by adding an edge e between them. By Lemma 5(iii), the increase in cost in the first step is at most $2\left(\hat{c}(C_r) + \hat{c}(T^*_{C_r})\right)$. Since $e \in G[V_2]$, from the triangle inequality $\hat{c}(e) \leqslant 0.5\,\hat{c}(\sigma^*_{V_2}) \leqslant 0.5\,\hat{c}(\sigma^*)$ where the last inequality follows from Lemma 4. Similarly, from the triangle inequality and Lemma 4, $\hat{c}(x, a) \leqslant 0.5\,\hat{c}(\sigma^*)$. Therefore, in this case, $\hat{c}(H) \leqslant 2\hat{c}(T^*) + \hat{c}(e) + \hat{c}(x, a) \leqslant 3\hat{c}(\sigma^*)$. □

4.2.2. Algorithm when depth of H is two

Now, we consider the case when the depth of the component graph is two. In this case, the removal of cross-edges from the T^* will result in more than one component from $T^*[V_1]$ and $T^*[V_2]$, respectively. Let $C_1 = \{U_1^1, \ldots, U_1^l\}$ and $C_2 = \{U_2^1, \ldots, U_2^p\}$ be the components from $T^*[V_1]$ and $T^*[V_2]$, respectively. We first merge the components in C_1 and C_2 into a single component, respectively. After the merging, we find a Hamiltonian path in both the components. Then, the appropriate cross-edges from the MST are added to get the Hamiltonian tour.

Without loss of generality, we assume that the component graph H is rooted at the vertex corresponding to the component $U_1^1 \in C_1$. If a vertex in H corresponding to a component from $C_2 = \{U_2^1, \ldots, U_2^p\}$ has degree greater than 1, we refer to it as an internal vertex.

In H, we pick a branch from the root to a leaf corresponding to a component from C_1. Note that such a branch exists because the depth of H is two. Then, we merge the leaf with the root. If the internal vertex in the path from the root to the leaf has more than one leaf incident to it, we first merge the leaves to get a path on them. Then, we merge that path with the root. Next, we pick two independent cross-edges $e_1 = (p_1, p_2)$ and $e_2 = (q_1, q_2)$ present in T^*. Then, we find a Hamiltonian path $HP_1(p_1, q_1)$ by using the double-tree algorithm.

Next, we construct a Hamiltonian path $HP_2(p_2, q_2)$ in $G[V_2]$ by using Hoogeveen's algorithm [10]. Finally, the Hamiltonian tour is given as $HP_1(p_1, q_1) \cup HP_2(p_2, q_2) \cup \{e_1, e_2\}$. The algorithm is presented in Algorithm 4. An illustration of the algorithm is shown in Fig. 5.

Before describing the algorithm, we explain a procedure called `CONNECT`(U_2^j), where U_2^j is an internal vertex of the component graph H. The procedure merges the component corresponding to the leaf incident to U_2^j with U_1^1. If U_2^j has more than one leaf, then it merges the components corresponding to the leaves in a path-like fashion and then attaches this path to U_1^1.

Lemma 9. *Let U_2^j be an internal vertex of H with more than one child. Let T be the Steiner tree of U_2^j rooted at r. Let $(r, \ldots, l)$ be the vertices in the pre-order traversal of T. Let $T_c^*(U_2^j)$ be the cross-edges of T^* incident to U_2^j. Let e_r and e_l be the cross-edges incident to the vertices r and l of T. Then, the sum of cost of edges used to merge the components incident to U_2^j by applying the procedure `CONNECT`(U_2^j) is at most $2\left(\hat{c}(U_2^j) + \hat{c}(T_c^*(U_2^j))\right) - \left(\hat{c}(P_T(r, l)) + \hat{c}(e_r) + \hat{c}(e_l)\right)$.*

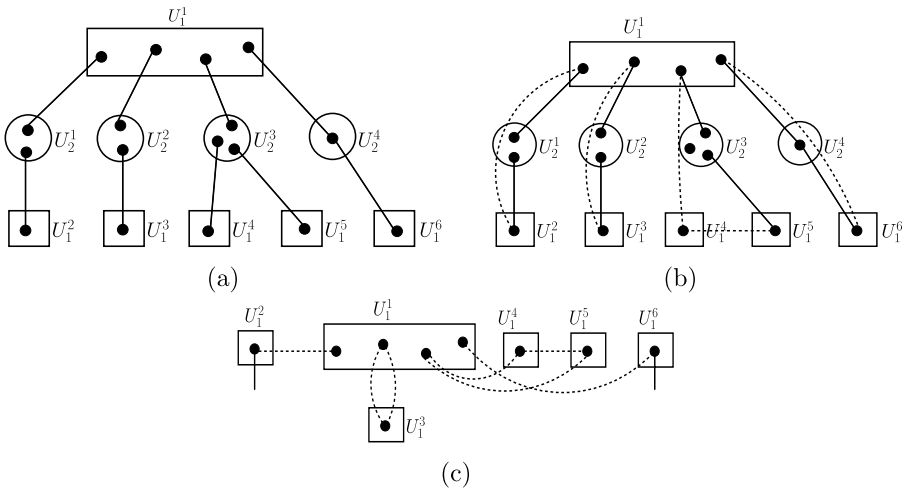


Fig. 5. Illustration of Phase 1 of Algorithm 4. (a) Component graph with $depth = 2$. (b) Merging of the leaf components with the root. The dotted lines indicate the edges used to merge the components. (c) Eulerian path between the components U_1^2 and U_1^6. The solid lines indicate the cross-edges.

Procedure: CONNECT(U_2^j).

Compute the Steiner tree T in the component U_2^j with respect to the marked vertices of U_2^j;

Root T at the marked vertex of T upon which the cross-edge connecting U_2^j to the component U_1^1 corresponding to the root is incident to;

Order the vertices of T in the pre-order traversal sequence of T (Note that T might not be a binary tree);

Let $U_1^i, U_1^{i+1}, \ldots, U_1^{i+k}$ be the ordered components in C_1 corresponding to the leaves of H as they are connected to the marked vertices in the pre-order traversal sequence of T;

For $i \leqslant q \leqslant i+k-1$, merge the components U_1^q, U_1^{q+1} by adding the edge (v_1^q, v_1^{q+1}) where v_1^q and v_1^{q+1} are the marked vertices in these two components. Let us call this merged component as C_1^j;

Let r_1^j be the marked vertex in the component U_1^1 upon which the cross-edge connecting U_1^1 to U_2^j is incident to. Merge C_1^j with U_1^1 by adding the edge (r_1^j, v_1^i);

return $U_1^1 \cup C_1^j \cup \{(r_1^j, v_1^i)\}$.

Proof. Let U_1^x and U_1^y be the component(s) from C_1 connected to the marked vertices x and y in T by cross-edges e_x and e_y. If U_1^x and U_1^y are merged, then by Lemma 1 the cost of the edge used to merge U_1^x and U_1^y is at most $\hat{c}(P_T(x,y)) + \hat{c}(e_x) + \hat{c}(e_y)$. The pre-order traversal of T visits all the edges twice except the edges on the path $P_T(r,l)$ which are visited exactly once. Since the components of C_1 connected to the marked vertices of T are merged in the pre-order traversal sequence of the vertices, the sum of cost of the edges used to merge the components is at most $2\left(\hat{c}(T) + \hat{c}(T_c^*(U_2^j))\right) - \left(\hat{c}(P_T(r,l)) + \hat{c}(e_r) + \hat{c}(e_l)\right)$. Since $T \subseteq U_2^j$, the sum of cost of the edges is bounded above by $2\left(\hat{c}(U_2^j) + \hat{c}(T_c^*(U_2^j))\right) - \left(\hat{c}(P_T(r,l)) + \hat{c}(e_r) + \hat{c}(e_l)\right)$. □

Lemma 10. *Two independent cross-edges are guaranteed to exist in Phase 1b of Algorithm 4.*

Proof. We first observe that, the degree of the root in H should be greater than 1. If it is not, we can root the component graph at the internal vertex and reduce $depth(H)$ by 1. Let H' be the graph obtained at the end of Phase 1a by Algorithm 4.

If H has only one internal vertex, then we can pick a cross-edge e_1 incident to the leaf of H which is also a leaf in H'. Since the root of H has degree at least 2, we can pick a cross-edge e_2 incident to a leaf of H which is not an internal vertex. It is clear that e_1 and e_2 are independent.

If H has more than one internal vertices, then we can pick two cross-edges e_1 and e_2 incident to the leaves of H which are also leaves in H'. It is easy to see that the leaves upon which e_1 and e_2 are incident are distinct. Also, the internal vertices upon which e_1 and e_2 are incident are distinct. Hence, e_1 and e_2 are independent. □

Lemma 11. *The cost of the Hamiltonian path HP_1 produced by Algorithm 4 at the end of Phase 2 along with the two independent cross-edges $e_1 = (v_1^1, u_2^1)$ and $e_2 = (v_1^2, u_2^2)$ is at most $2\,\hat{c}(\sigma^*)$.*

Algorithm 4: Algorithm for 1-MST whose component graph has $depth = 2$.

Input: 1-MST T^* and its component graph H with $depth(H) = 2$.
Output: A Hamiltonian tour.
Phase 1a
Let $C_1 = \{U_1^1, \ldots, U_1^l\}$ and $C_2 = \{U_2^1, \ldots, U_2^p\}$ be the components of $T^*[V_1]$ and $T^*[V_2]$ after the removal of cross-edges from T^*;
$H' = \emptyset$;
while *the components of* C_1 *are not connected* **do**
  Pick an internal vertex U_2^j of H;
  $H' = H' \cup$ CONNECT(U_2^j);
  From H remove U_2^j and the child (children) of U_2^j;
end
Phase 1b
Pick two independent cross-edges $e_1 = (v_1^1, u_2^1)$ and $e_2 = (v_1^2, u_2^2)$ incident to the leaves of H (Note that leaves in H are chosen such that they are also leaves in H');
Phase 2
for *each internal vertex* $U_2^j \in C_2$ *with more than one child* **do**
  Let $C_1^j = (U_1^1, \ldots, U_1^i)$ be the components of C_1 appearing in the given order obtained by applying the procedure CONNECT(U_2^j);
  if v_1^1 *and* $v_1^2 \notin$ *children of* U_2^j **then**
    Add an edge e between the marked vertices of the components U_1^1 and U_1^i. $H' = H' \cup \{e\}$;
  end
end
Double all the edges in H' except the following:
1. The edges on the path between v_1^1 and v_1^2.
2. The edges used to merge the components by applying procedure CONNECT(U_2^k) where U_2^k is an internal vertex in H with more than one child.

Obtain an Eulerian path in H' between v_1^1 and v_1^2. Short-cut it to get a Hamiltonian path $HP_1(v_1^1, v_1^2)$;
Phase 3
Construct a Hamiltonian path $HP_2(u_2^1, u_2^2)$ between u_2^1 and u_2^2 in $G[V_2]$ using Hoogeveen's algorithm (Lemma 3);
Construct the tour $\sigma = HP_1(v_1^1, v_1^2) \cup HP_2(u_2^1, u_2^2) \cup \{e_1, e_2\}$;
return σ.

Proof. Let H' be the multigraph obtained at the end of Phase 2 of Algorithm 4. It is easy to see that H' has an Eulerian path between v_1^1 and v_1^2. For any component $U_2^j \in C_2$, let us denote the cross-edges incident to U_2^j as $T_c^*(U_2^j)$.

For an internal vertex U_2^j that has more than one leaf incident to it, let $U_1^1, \ldots, U_1^i$ be the components merged in this order by applying the procedure CONNECT(U_2^j). Hence, from Lemma 9, the cost of the edges used to merge the components $U_1^1, \ldots, U_1^i$ is at most $2\left(\hat{c}(U_2^j) + \hat{c}(T_c^*(U_2^j))\right) - \left(\hat{c}(P_T(r,l)) + \hat{c}(e_r) + \hat{c}(e_l)\right)$ where $P_T(r,l)$ is the path between r and l in the pre-order traversal $(r, \ldots, l)$ of the Steiner tree T of U_2^j, and e_r and e_l are the cross-edges incident to the vertices r and l. If both e_1 and e_2 are not incident to any of the components $U_1^1, \ldots, U_1^i$, then Algorithm 4 adds an edge between the marked vertices of U_1^1 and U_1^i. Since U_1^1 and U_1^i are connected by cross-edges to the vertices r and l in T, by Lemma 1, the cost of the edge used to merge U_1^1 and U_1^i is at most $\hat{c}(P_T(r,l)) + \hat{c}(e_r) + \hat{c}(e_l)$. Hence, the sum of cost of the edges used to merge the components incident to U_2^j is at most $2\left(\hat{c}(U_2^j) + \hat{c}(T_c^*(U_2^j))\right)$.

If the internal vertex U_2^j has only one leaf incident to it, then the component U_1^j corresponding to the leaf is merged with the component U_1^1 by adding an edge between their marked vertices. By Lemma 1, the cost of the edge is at most $\hat{c}(U_2^j) + \hat{c}(T_c^*(U_2^j))$. Note that if e_1 and e_2 are not incident to U_1^j, then two copies of this merged edge will be used in the final graph H'.

Since the paths between the root to the internal vertices are all disjoint, the cost of the edges in H' (including multiple copies) used to merge the components from C_1 is at most $2\sum_{j=1}^{p}\left(\hat{c}(U_2^j) + \hat{c}(T_c^*(U_2^j))\right) - \hat{c}(e_1) - \hat{c}(e_2)$. Since H' contains two copies of all the edges in the components of C_1, the cost of the Eulerian path between v_1^1 and v_1^2 in H' is at most $2\left(\sum_{j=1}^{l}\hat{c}(U_1^j) + \sum_{j=1}^{p}\left(\hat{c}(U_2^j) + \hat{c}(T_c^*(U_2^j))\right)\right) - \hat{c}(e_1) - \hat{c}(e_2) = 2\hat{c}(T^*) - \hat{c}(e_1) - \hat{c}(e_2)$.

Since the Eulerian path is completely contained in the vertex set V_1, we can short-cut the Eulerian path to get a Hamiltonian path $HP_1(v_1^1, v_1^2)$ without increasing the cost. Therefore, $\hat{c}(HP_1(v_1^1, v_1^2)) + \hat{c}(e_1) + \hat{c}(e_2) \leqslant 2\hat{c}(T^*) \leqslant 2\hat{c}(\sigma^*)$. □

Lemma 12. *Algorithm 4 is a 3.5-approximation algorithm for the* Biased-TSP.

Proof. $\sigma = HP_1(v_1^1, v_1^2) \cup HP_2(u_2^1, u_2^2) \cup \{e_1, e_2\}$ is the Hamiltonian tour returned by Algorithm 4. From Lemma 11, $\hat{c}(HP_1(v_1^1, v_1^2)) + \hat{c}(e_1) + \hat{c}(e_2) \leqslant 2\hat{c}(\sigma^*)$. By Lemma 3 and Lemma 4, $\hat{c}(HP_2(u_2^1, u_2^2)) \leqslant 1.5\hat{c}(\sigma^*)$. Hence, the statement of the theorem follows. □

4.2.3. Algorithm when depth of H is greater than two

Without loss of generality, let us assume that H is rooted at $U_1^1 \in C_1$. In this case, we first merge all the components of C_1 by iteratively picking subtrees of $depth = 2$ from the bottom of the tree and then applying Phase 1a of Algorithm 4 to them. Then, using Phase 1b of Algorithm 4 we pick two independent cross-edges. Finally, we obtain a Hamiltonian tour using Phase 3 of Algorithm 4. The algorithm is presented in Algorithm 5.

In order to merge all the components of C_1 into a single component, we first pre-process H by deleting the leaves of H corresponding to the components from C_2. After this all the leaves of H will be from C_1 and hence, we can iteratively apply Phase 1a of Algorithm 4 till all the components from C_1 are connected. Now, we select a leaf of H which is at a maximum distance from U_1^1. Then, we select a vertex U_1^j of H which is at a distance of two from the leaf on the path between the leaf and the root. We pick the subtree rooted at U_1^j and apply Phase 1a of Algorithm 4 to it. Then, we remove all the internal vertices and the leaves of the subtree. We perform this procedure till all the components from C_1 are connected.

Algorithm 5: Algorithm for 1-MST with $depth > 2$.

Input: 1-MST T^* and its component graph H with $depth(H) > 2$.
Output: A Hamiltonian tour.
Delete all the leaves of H which correspond to the components from C_2;
$H'' = \emptyset$;
while *the components from* C_1 *are not connected* **do**
  Pick a leaf of H which is at a maximum distance from the root;
  On the path between the leaf and the root pick a vertex U_1^j which is at a distance of 2 from the leaf;
  Run Phase 1a of Algorithm 4 on the subtree rooted at U_1^j;
  Let H' be the output produced by Phase 1a of Algorithm 4;
  $H'' = H'' \cup H'$;
end
Run Phase 1b of Algorithm 4 on the last subtree of H to obtain two independent cross-edges $e_1 = (v_1^1, u_2^1)$ and $e_2 = (v_1^2, u_2^2)$;
Run Phase 2 of Algorithm 4 on H'' to obtain a Hamiltonian path $HP_1(v_1^1, v_1^2)$ on V_1;
Run Phase 3 of Algorithm 4 to obtain a Hamiltonian tour σ;
return σ.

Lemma 13. *Algorithm 5 is 3.5 approximation algorithm for the* `Biased-TSP`.

Proof. First, we observe that the root of the component graph H has degree at least two. Hence, from Lemma 10, we know that, we can pick two independent cross-edges $e_1 = (v_1^1, u_2^1)$ and $e_2 = (v_1^2, u_2^2)$ from the final subtree of H.

The while loop in Algorithm 5 repeatedly applies Phase 1a of Algorithm 4 till all the components from C_1 are merged to form a single component H''. Hence, by applying Phase 2 of Algorithm 4 on H'', we can obtain an Eulerian path between v_1^1 and v_1^2 spanning the vertices of V_1. Since this Eulerian path is completely contained within V_1, it can be short-cut to obtain a Hamiltonian path without increase in the cost. Since the internal vertices of the subtree of H do not overlap, by Lemma 11, the cost of the Hamiltonian path HP_1 on V_1 along with the two independent cross-edges e_1 and e_2 is at most $2\hat{c}(\sigma^*)$. The cost of Hamiltonian path HP_2 on $G[V_2]$ is at most $1.5\hat{c}(\sigma^*)$.

Hence, the cost of the tour $\sigma = HP_1 \cup HP_2 \cup \{e_1, e_2\}$ is no greater than 3.5 times the cost of the optimal solution. □

From Lemma 8, Lemma 12, and Lemma 13, we get the following.

Theorem 4. *If an instance of the* `Biased-TSP` *has a 1-MST, then it can be approximated within a factor of 3.5.*

From Theorem 3 and Theorem 4, we get the following.

Theorem 5. *The* `Biased-TSP` *can be approximated within a factor of 3.5.*

5. Conclusion

We proved that `Biased-TSP` can be approximated within a factor of 3.5 which is independent of the bias factor β. However, when $\beta = 1$, `Biased-TSP` is the same as $\triangle$-TSP and can be approximated within a factor of 1.5. Therefore, it would be interesting to obtain an approximation algorithm with approximation factor of $f(\beta)$ (here β is a constant for all the cross-edges) with $f(1) = 1.5$. Other interesting question would be to study the approximability properties of `Biased-TSP` when the vertex set is partitioned into more than two parts.

Acknowledgement

The authors thank the referees for their valuable comments and suggestions which helped revise and reorganize this paper.

References

[1] T. Andreae, On the traveling salesman problem restricted to inputs satisfying a relaxed triangle inequality, Networks 38 (2) (2001) 59–67.
[2] T. Andreae, H.J. Bandelt, Performance guarantees for approximation algorithms depending on parametrized triangle inequalities, SIAM J. Discrete Math. 8 (1) (1995) 1–16.
[3] D.L. Applegate, R.E. Bixby, V. Chvátal, W.J. Cook, The Traveling Salesman Problem: A Computational Study, Princeton University Press, 2011.
[4] S. Arora, Polynomial time approximation schemes for Euclidean traveling salesman and other geometric problems, J. ACM 45 (5) (1998) 753–782.
[5] M.A. Bender, C. Chekuri, Performance guarantees for the TSP with a parameterized triangle inequality, Inform. Process. Lett. 73 (1) (2000) 17–21.
[6] H.J. Böckenhauer, J. Hromkovič, R. Klasing, S. Seibert, W. Unger, Towards the notion of stability of approximation for hard optimization tasks and the traveling salesman problem, Theoret. Comput. Sci. 285 (1) (2002) 3–24.
[7] H.J. Böckenhauer, L. Forlizzi, J. Hromkovič, J. Kneis, J. Kupke, G. Proietti, P. Widmayer, On the approximability of TSP on local modifications of optimally solved instances, Algorithmic Oper. Res. 2 (2007) 83–93.
[8] N. Christofides, Worst-case analysis of a new heuristic for the travelling salesman problem, Technical Report 388, Graduate School of Industrial Administration, Carnegie-Mellon University, 1976.
[9] M.R. Garey, D.S. Johnson, Computers and Intractability: A Guide to the Theory of NP-completeness, Freeman, San Francisco, LA, 1979.
[10] J.A. Hoogeveen, Analysis of Christofides' heuristic: some paths are more difficult than cycles, Oper. Res. Lett. 10 (5) (1991) 291–295.
[11] J.S.B. Mitchell, Guillotine subdivisions approximate polygonal subdivisions: a simple polynomial-time approximation scheme for geometric TSP, k-MST, and related problems, SIAM J. Comput. 28 (4) (1999) 1298–1309.
[12] C.H. Papadimitriou, K. Steiglitz, Combinatorial Optimization: Algorithms and Complexity, Prentice-Hall, Englewood Cliffs, NJ, 1982.
[13] S. Sahni, T. Gonzalez, P-complete approximation problems, J. ACM 23 (3) (1976) 555–565.
[14] A. Shurbevski, H. Nagamochi, Y. Karuno, Approximating the Bipartite TSP and its Biased Generalization, in: Proceedings of WALCOM, in: Lecture Notes in Computer Science, vol. 8344, 2014, pp. 56–67.
[15] D.B. West, Introduction to Graph Theory, Prentice-Hall, Upper Saddle River, 2001.