Approximating bottleneck spanning trees on partitioned
tuples of points

Ahmad Biniaz† Anil Maheshwari‡ Michiel Smid†

Abstract

We present approximation algorithms for the following NP-hard optimization problems related to bottleneck spanning trees in metric spaces.

1. The disjoint bottleneck spanning tree problem: Given \( n \) pairs of points in a metric space, find two disjoint trees each containing exactly one point from each pair and minimize the largest edge length (over all edges of both trees). It is known that approximating this problem by a factor better than 2 is NP-hard. We present a 4-approximation algorithm for this problem. This improves upon the previous best known approximation ratio of 9. Our algorithm extends to a \((3k - 2)\)-approximation for a more general case where points are partitioned into \( k \)-tuples and we seek \( k \) disjoint trees.

2. The generalized bottleneck spanning tree problem: Given \( n \) points in some metric space that are partitioned into clusters of size at most 2, find a tree that contains exactly one point from each cluster and minimizes the largest edge length. We show that it is NP-hard to approximate this problem by a factor better than 2, and present a 3-approximation algorithm.

3. The partitioned bottleneck spanning tree problem: Given \( kn \) points in some metric space, find \( k \) trees each containing exactly \( n \) points and minimize the largest edge length (over all edges of the \( k \) trees). We show that it is NP-hard to approximate this problem by a factor better than 2 for any \( k \geq 2 \). We present an \( \alpha \)-approximation algorithm for this problem where \( \alpha = 2 \) for \( k = 2,3 \) and \( \alpha = 3 \) for \( k \geq 4 \). Towards obtaining these approximation ratios we present tight upper bounds on the edge lengths of \( k \) equal-size disjoint trees that can be obtained from the nodes of a given tree. This result is of independent interest.

Our hardness proofs imply that it is NP-hard to approximate the non-metric version of the above problems within any constant factor. If we seek traveling salesperson tours (instead of trees) then our algorithms simply extend to achieve approximate solutions with factors three times those mentioned above.

1 Introduction

Spanning tree is a fundamental structure in graph theory and combinatorics. The problem of finding spanning trees with enforced properties has received considerable attention from both theoretical and practical points of view. For example, the minimum spanning tree (MST) problem asks for a spanning tree with minimum total edge-length, and the bottleneck spanning tree (BST) problem asks for a spanning tree whose largest edge-length is minimum. Beside their interesting theoretical
properties, these problems find applications in the design of networks, including computer networks, wireless networks, and transportation networks, to name a few. Bottleneck spanning trees in particular are important in designing telecommunications networks with short connections (edges). Short connections are desirable in many ways because they require lower transmission ranges, are more secure, and cause less interference. This paper addresses three closely related bottleneck spanning tree problems (illustrated in Figure 1):

1. The **disjoint bottleneck spanning tree** (k-DBST) problem: Given \( kn \) points in some metric space that are partitioned into \( k \)-tuples, find \( k \) disjoint trees each containing exactly one point from each tuple and minimize the largest edge length (over all edges of the \( k \) trees).

2. The **generalized bottleneck spanning tree** (k-GBST) problem: Given \( n \) points in some metric space that are partitioned into clusters of size at most \( k \), find a tree that contains exactly one point from each cluster and minimizes the largest edge length. The term “spanning” refers to span all clusters.

3. The **partitioned bottleneck spanning tree** (k-PBST) problem: Given \( kn \) points in some metric space, find \( k \) trees each containing exactly \( n \) points and minimize the largest edge length (over all edges of the \( k \) trees).

The above problems are natural generalizations of the standard BST problem. For \( k = 1 \), all above problems are equivalent to the BST problem which can be solved optimally in polynomial time [9]. For \( k \geq 2 \), all above problems are NP-hard and cannot be approximated by a factor better than 2 unless P = NP (this will become clear shortly). The focus of this paper is on \( k \geq 2 \).

We first present constant-factor approximation algorithms for \( k = 2 \). Then we extend some of our algorithms for larger \( k \).

### 1.1 Some related works and applications

The problems introduced above find real-world applications that we put into context together with some related works. In our description we implicitly assume that \( k \) is at least 2.

1. The k-DBST problem is introduced by Arkin et al. [2]. Motivated by the problem of maintaining secure connectivity in networks involving replicated data, Arkin et al. [2] introduced a class of problems that ask for \( k \) disjoint structures (trees, cycles, matchings) each containing one point form every given \( k \)-tuple. In particular they studied these problems for \( k = 2 \). Among many interesting results they presented a 9-approximation algorithm for the 2-DBST problem and an 18-approximation algorithm for computing two disjoint traveling salesperson tours (instead of trees). It is easily seen, from their Lemma 8, that the 9-approximation algorithm can be extended to achieve
a $(6k − 3)$-approximation for the $k$-DBST problem. Although some of the results of Arkin et al. [2] have been improved by Johnson [21], their ratios 9 and 18 are still the best known. As for the lower bound, Johnson [21] showed that it is NP-hard to approximate the 2-DBST problem by a factor better than 2.

(2) The $k$-GBST problem is closely related to the $k$-generalized minimum spanning tree ($k$-GMST) problem, introduced by Myung et al. [29]. The $k$-GMST problem asks for a tree that contains exactly one point from each cluster and minimizes the total-edge length. This problem is well studied (see e.g. the recent survey by Pop [34] and references therein). The $k$-GMST problem is NP-hard even for $k = 2$ in the Euclidean plane. Even a more restricted version where the two points in each cluster have the same $x$ or $y$ coordinates is NP-hard [13, 16, 22]. The metric version of the $k$-GMST can be approximated by a ratio of $2k$ using linear programming [36] combined with the so-called parsimonious property [18]. Related work [6, 34, 35] also addresses the generalized traveling salesperson problem (TSP) in which the tour must contain exactly one point from each cluster. The group Steiner tree is another related problem which asks for a shortest tree that contains at least one point from each cluster. The non-metric versions of both the $k$-GMST and the group Steiner tree problems are NP-hard and cannot be approximated within any constant factor [20, 29]. Gabow et al. [17] studied the problem of finding a path, from a source to a destination in a graph, that passes through at most one vertex from every given pair of vertices. Arkin et al. [3] studied the multiple-choice minimum-diameter problem which is to select at least one element from each cluster to minimize the diameter of the chosen elements. The $k$-GBST also lies in the concept of imprecision in computational geometry where each input point is provided as a region of uncertainty (also known as neighborhood) and the exact position of the point may be anywhere in the region; see e.g. [7, 14, 26, 27, 28].

Both the $k$-GBST and the $k$-GMST have real-world applications for example in the field of telecommunications, designing metropolitan area networks, interconnecting local area networks, determining location of regional service centers (e.g., stores, warehouses, agricultural settings, distribution centers). For a detailed explanation of these applications and for more examples we refer the interested reader to the paper of Myung et al. [29] and the recent survey by Pop [34].

(3) The $k$-PBST problem falls in the class of partitioning a set into subsets such that the substructures (computed on subsets) are balanced. Balanced partitioning of the input has a long history and gives rise to interesting theoretical problems. For example in the $k$-partition traveling salesperson problem we are given $k$ salespersons and the goal is to visit every city by exactly one salesperson and minimize the distance traveled by the salesperson making the longest journey [4, 5, 33].

The problem of $k$-balanced partitioning of a graph asks for partitioning the vertices of the graph into $k$ subsets such that the induced subgraph on each subset is connected and the maximum cardinality of the subsets is minimized. Dyer and Frieze [15] showed that this problem is NP-hard; they also showed the hardness of many variations of this problem. Chlebíková [11] presented constant-factor approximations for $k = 2, 3$, and Chen et al. [10] presented a $k/2$-approximation for $k \geq 4$. The max-min version of this problem is also studied [11, 38].

Motivated by a problem from the shipbuilding industry, Andersson et al. [1] studied the $k$-partition minimum spanning tree ($k$-PMST) problem where the goal is to partition an input point set into $k$ subsets such that the length of the longest MST on the subsets is minimized. As noted in [24] (and references therein) the $k$-PMST problem also arises in multi-vehicle scheduling, task sequencing, and political districting. Andersson et al. [1] showed that the $k$-PMST problem is NP-hard even for $k = 2$ in the Euclidean metric in the plane, and presented $(4/3 + \epsilon)$ and $(2 + \epsilon)$
approximations for $k = 2$ and $k \geq 3$, respectively. Karakawa et al. [24] studied this problem in higher dimensions. The $k$-PMST problem has also been studied in trees and cactus graphs under the name “minmax subtree cover” problem [30, 31, 32].

1.2 Our contributions

We study the $k$-DBST, $k$-GBST, and $k$-PBST problems in metric spaces (where distances satisfy the triangle inequality). We show the hardness as well as approximation algorithms for these problems. We present our results for the simplest version where $k = 2$ (as it is easier to understand) and then extend them for larger $k$.

- The 2-DBST problem is known [21] to be NP-hard and inapproximable by a factor better than 2. We present a 4-approximation algorithm for this problem. This improves the previous best known ratio of 9 due to Arkin et al. [2]. We extend our algorithm and achieve a $(3k - 2)$-approximation for the $k$-DBST for any $k \geq 2$ (Theorem 3).

- The difficulty of the 2-GBST problem lies in choosing representative points from clusters; once these points are selected, the problem is reduced to the standard BST problem. We show that it is NP-hard to approximate the 2-GBST problem by a factor better than 2 using a reduction from 3-SAT (Theorem 4), and present a 3-approximation algorithm for this problem (Theorem 5). In some part of our algorithm we show the following result which is of independent interest (Theorem 6): Given a tree $T_1$ and a partitioning of its nodes into clusters of size at most two, we can obtain a tree $T_2$ that contains exactly one node from each cluster and the length of its edges is at most 3 in the metric\(^1\) of $T_1$; the upper bound 3 is the best achievable.

- We show that it is NP-hard to approximate the $k$-PBST problem by a factor better than 2 for any $k \geq 2$ (Theorem 7) using a reduction from the 2-balanced partitioning of a graph [15]. We present an $\alpha$-approximation algorithm for this problem (Theorem 8) where $\alpha = 2$ for $k = 2, 3$ and $\alpha = 3$ for $k \geq 4$. Towards obtaining these approximation ratios we present tight upper bounds on the edge lengths of $k$ equal-size disjoint trees that can be obtained from the nodes of a given tree (Theorem 9). This result is of independent interest.

A straightforward implication of our hardness proofs and that of Johnson [21] is that the non-metric versions of the above problems cannot be approximated within any constant factor.

Extension to bottleneck TSP tours. If instead of trees in the above problems we seek TSP tours that minimize the largest edge length, then our algorithms simply extend to obtain approximate solutions with factors that are thrice those for bottleneck trees. This can be done via a known result that the cube\(^2\) of every connected graph has a Hamiltonian cycle, and such a cycle can be computed in polynomial time [23, 25]; this is also hinted in [12, Exercise 37.2.3]. To use this result, we first obtain an $\alpha$-approximate solution, namely $B$, for the corresponding BST problem (using our BST algorithms) and then we find TSP tours, namely $T$, in the cube of $B$. By the triangle inequality the largest edge-length in the cube graph, and in particular in $T$, is at most thrice the largest edge-length in $B$. Notice that in all above problems the largest edge length in any optimal

\(^1\)In this metric the distance between two nodes $u$ and $v$ in a tree $T$ is the number of edges in the unique path between them in $T$.

\(^2\)The cube of a graph $G$ has the same vertices as $G$, and has an edge between two distinct vertices if and only if there exists a path, with at most three edges, between them in $G$. 
A BST solution is a lower bound for the largest edge length in any optimal TSP solution. Thus 𝑇 would be a 3𝑛-approximate solution for the TSP. For example our 4-approximation algorithm for the 2-DBST can be extended to obtain a 12-approximation for two disjoint TSP tours that minimize the largest edge length; this improves the previous approximation ratio of 18 due to Arkin et al. [2].

Notation. The largest edge length in a tree 𝑇 is referred to as the bottleneck of 𝑇 and is denoted by 𝜆(𝑇). We denote the distance between two points 𝑝 and 𝑞 in a metric space by [𝑝𝑞]. Conceptually, a point set 𝑃 in a metric space can be viewed as a metric graph, i.e., as a complete edge-weighted graph with vertex set 𝑃 where the weight 𝑤(𝑒) of each edge 𝑒 = (𝑝, 𝑞) is equal to the distance between 𝑝 and 𝑞, that is 𝑤(𝑒) = [𝑝𝑞].

2 The 𝑘-DBST problem

Let 𝑘 ≥ 2 be an integer. In this section we present an approximation algorithm for the 𝑘-DBST problem: Given 𝑘𝑛 points in some metric space that are partitioned into 𝑘-tuples, we want to find 𝑘 disjoint trees each containing exactly one point from each tuple and minimize the largest edge length (over all the 𝑘 trees). We first present our approximation algorithm for 𝑘 = 2 as it is easier to understand. Then we extend the algorithm to larger 𝑘. Our algorithm benefits from the following remarkable result of König which is stated in [19].

Theorem 1 (König, 1916). Let 𝑆 be any set with 𝑘𝑛 elements that is partitioned, in two different ways, into 𝑛 subsets each with 𝑘 elements, namely 𝑎1, . . . , 𝑎𝑛 and 𝑏1, . . . , 𝑏𝑛. Then there exist 𝑛 elements of 𝑆, namely 𝑟1, . . . , 𝑟𝑛, and a permutation 𝜋 of {1, . . . , 𝑛} such that 𝑟𝑖 ∈ 𝑎𝑖 ∩ 𝑏 𝜋 𝑖 for all 𝑖 ∈ {1, . . . , 𝑛}.

Example. Let 𝑘 = 3, 𝑛 = 4, 𝑆 = {1, 2, . . . , 12}, and consider two partitions of 𝑆

| 𝑎1 | 𝑎2 | 𝑎3 | 𝑎4 |
|---|---|---|---|
| 1, 2, 3 | 4, 5, 6 | 7, 8, 9 | 10, 11, 12 |

| 𝑏1 | 𝑏2 | 𝑏3 | 𝑏4 |
|---|---|---|---|
| 4, 9, 12 | 2, 8, 11 | 1, 3, 5 | 6, 7, 10 |

Then by taking 𝑟1 = 1, 𝑟2 = 6, 𝑟3 = 8, 𝑟4 = 12, and 𝜋 = (3, 4, 2, 1) we get that

| 𝑟1 | 𝑟2 | 𝑟3 | 𝑟4 |
|---|---|---|---|
| 1 ∈ 𝑎1 ∩ 𝑏3 | 6 ∈ 𝑎2 ∩ 𝑏4 | 8 ∈ 𝑎3 ∩ 𝑏2 | 12 ∈ 𝑎4 ∩ 𝑏1 |

Hall (1935) showed a more general version of König’s theorem (where subsets can have different sizes) as an implication of his famous result [19]—today known as the Hall’s marriage theorem. The set 𝑅 = {𝑟1, . . . , 𝑟𝑛} in Theorem 1 is called a complete system of representatives for subsets 𝑎𝑖 (and also for subsets 𝑏𝑖). The following theorem (which is a generalized version of Lemma 8 in [2]) is an implication of König’s theorem.

Theorem 2. Let 𝑆 be a set with 𝑘𝑛 elements that is partitioned, in two different ways, into 𝑛 subsets each with 𝑘 elements, namely 𝑎1, . . . , 𝑎𝑛 and 𝑏1, . . . , 𝑏𝑛. Then, it is possible to label all elements of 𝑆 with 𝑘 distinct labels such that the 𝑘 elements in each of 𝑎1, . . . , 𝑎𝑛, 𝑏1, . . . , 𝑏𝑛 have 𝑘 distinct labels. Moreover, such a labeling can be found in polynomial time.

Proof. By König’s theorem there exists a subset 𝑅 = {𝑟1, . . . , 𝑟𝑛} of 𝑆 that is a complete system of representatives for subsets 𝑎𝑖 and for subsets 𝑏𝑖. Such a system 𝑅 can be found as follows. Construct a bipartite graph 𝐺 = (𝑉, 𝐸) with 2𝑛 vertices such that 𝑉 = {𝑎1, . . . , 𝑎𝑛, 𝑏1, . . . , 𝑏𝑛} and there is an edge between 𝑎𝑖 and 𝑏𝑗 if and only if 𝑎𝑖 ∩ 𝑏𝑗 ≠ ∅. According to Hall’s marriage theorem [8, 19] 𝐺 has a perfect matching 𝑀 (with 𝑁 edges) which can be found in polynomial.
time. For every edge \((A_i, B_j)\) in \(M\) pick an arbitrary representative element in \(A_i \cap B_j\). These \(n\) representatives form \(R\).

Label all elements of \(R\) by \(l_1\). Then remove the vertices of \(R\) from \(S\) and from corresponding subsets \(A_i\) and \(B_j\). As a result we obtain a new set \(S\) with \((k - 1)n\) elements and two distinct partitions of \(S\) each with \(n\) subsets of size \(k - 1\). By applying König’s and Hall’s theorems we can find another complete system of representatives, and label them \(l_2\). Repeating the above process achieves a desired labeling \(l_1, \ldots, l_k\).

In the example above we can label elements of \(S\) by \(k = 3\) labels \(l_1, l_2, l_3\) where (with a slight abuse of notation) \(l_3 = \{1, 6, 8, 12\}, l_2 = \{2, 5, 9, 10\}, \) and \(l_3 = \{3, 4, 7, 11\}\) such that all elements in each \(A_i\) and \(B_i\) have different labels.

### 2.1 A 4-approximation for the 2-DBST

In this section we present a 4-approximation algorithm for the 2-DBST problem. Let \(P\) be a set of \(2n\) points in a metric space that is partitioned into \(n\) tuples \(A_1, \ldots, A_n\) each with two points. Let \(\lambda^*\) denote the bottleneck of a fixed optimal solution (consisting of two trees). We show how to find two disjoint trees \(R\) and \(B\) with edges of length at most \(4\lambda^*\). To simplify our description we assume that the nodes of \(R\) and \(B\) are colored red and blue, respectively.

We start by computing a minimum spanning tree of \(P\), which is also a bottleneck spanning tree. Let \(e\) be a longest edge of \(T\), that is \(\lambda(T) = w(e)\). Let \(T_1\) and \(T_2\) be the two trees obtained by removing \(e\) from \(T\). Notice that \(\max\{\lambda(T_1), \lambda(T_2)\} \leq w(e)\). If each \(A_i\) has a point in \(T_1\) and a point in \(T_2\), then we claim that \(R = T_1\) and \(B = T_2\) form an optimal solution because if the fixed optimal solution contains an edge between a node of \(T_1\) and a node of \(T_2\) then the length of that edge is at least \(w(e)\) which implies that \(\lambda^* \geq w(e)\). Therefore \(\max\{\lambda(R), \lambda(B)\} \leq \lambda^*\).

Now assume that both points of some tuple \(A_i\) belong to say \(T_1\). In any feasible solution, one point of \(A_i\) is red and the other is blue. Then regardless of the coloring of the nodes of \(T_2\), the optimal solution should contain an edge between a node of \(T_1\) and a node of \(T_2\). Thus \(\lambda^* \geq w(e)\). We are going to color the nodes of \(T\) (which are the points of \(P\)) red and blue and then obtain \(R\) and \(B\) in such a way that \(\max\{\lambda(R), \lambda(B)\} \leq 4 \cdot \lambda(T)\). This will imply that \(\max\{\lambda(R), \lambda(B)\} \leq 4\lambda^*\).

![Figure 2: (a) Creating buckets. (b) Construction of \(R\); dashed edges represent \(\delta_i\)](image)

We root \(T\) at a leaf \(q\). Then we partition the nodes of \(T\) into \(n\) buckets \(B_1, \ldots, B_n\) each with two vertices. The partitioning is done iteratively in a bottom-up fashion as follows. Consider a deepest leaf \(l\) and let \(v\) be its parent. Let \(u_1, u_2, \ldots, u_j\) be the children of \(v\) where \(u_1 = l\) as in Figure 2(a). If \(j\) is even then we create \(j/2\) buckets \(\{u_1, u_2\}, \{u_3, u_4\}, \ldots, \{u_{j-1}, u_j\}\), and then retain \(u_1, \ldots, u_j\) from \(T\). If \(j\) is odd then we create \((j + 1)/2\) buckets \(\{v, u_1\}, \{u_2, u_3\}, \{u_4, u_5\}, \ldots, \{u_{j-1}, u_j\}\), and then remove \(v, u_1, \ldots, u_j\) from \(T\). Then we repeat the above process until \(q\) and its only child form a bucket. We denote this last bucket by \(B_n\). The total number of buckets is \(n\) because \(T\) has \(2n\) nodes initially. Between any two nodes in the same bucket there exists a path of length at most 2 in \(T\), because the two nodes are either siblings or a child and its parent.
Now that we have two partitions $A_1, \ldots, A_n$ and $B_1, \ldots, B_n$ of $P$, we color (or label) the points of $P$ by two colors, red and blue, as in Theorem 2. Thus in each $A_i$ and each $B_i$ we get a red point and a blue point. We construct the tree $R$ by interconnecting the red points of buckets as follows; see Figure 2(b): Consider each bucket $B_i$ with $i \in \{1, \ldots, n - 1\}$ and let $r_i$ denote its red point.

(i) If the parent of $r_i$ is not in $B_i$, then we connect $r_i$ to the red point of its parent’s bucket.

(ii) If the parent of $r_i$ is in $B_i$, then we connect $r_i$ to the red point of its grandparent’s bucket.

We construct the tree $B$ on the blue points in a similar fashion. We claim that $R$ and $B$ are the desired trees. Since each $A_i$ contains a red point and a blue point (by Theorem 2), each of $R$ and $B$ contains exactly one point from $A_i$. Thus $R$ and $B$ form a feasible solution for the problem.

Analysis of the approximation ratio. We show that $\lambda(R) \leq 4 \cdot \lambda(T)$; an analogous argument holds for $B$. Root $R$ at the red point of $B_n$. Consider any red node $r_i$ in $R$ where $i \in \{1, \ldots, n - 1\}$. Recall that $r_i \in B_i$. Let $p_i$ be the parent of $r_i$ in $R$. It suffices to show that $|r_ip_i| \leq 4 \cdot \lambda(T)$.

Consider the unique path $\delta_i$ between $r_i$ and $p_i$ in $T$. See Figure 2(b). If $r_i$ was connected to $p_i$ in step (i) then $\delta_i$ has at most 3 edges. If $r_i$ was connected to $p_i$ in step (ii) then $\delta_i$ has at most 4 edges. Therefore $|r_ip_i| \leq w(\delta_i) \leq 4 \cdot \lambda(T)$.

2.2 A $(3k - 2)$-approximation for the $k$-DBST

Here we extend our 4-approximation algorithm of the previous section to get a $(3k-2)$-approximation for the $k$-DBST problem. We should note that (although it is not mentioned explicitly in their paper) Theorem 7 from Arkin et al. [2] combined with their Lemma 8 already gives a $(6k - 3)$-approximation for the $k$-DBST problem.

Let $P$ be a set of $kn$ points that is partitioned into $n$ tuples $A_1, \ldots, A_n$ each with $k$ points. Let $\lambda^*$ denote the bottleneck of a fixed optimal solution (consisting of $k$ trees). We show how to color the points in each $A_i$ by $k$ colors $c_1, \ldots, c_k$, and to obtain a tree $T_i$ on all points with color $c_i$ such that $\lambda(T_i) \leq (3k - 2)\lambda^*$.

Let $T$ be a minimum spanning tree of $P$. Root $T$ at a leaf $q$. We partition the nodes of $T$ into $n$ buckets $B_1, \ldots, B_n$ each with $k$ nodes. The partitioning is done iteratively in a bottom-up fashion. We describe it for obtaining bucket $B_j$. For each node $v$ in the current tree $T$, let $N(v)$ denote the number of nodes in the subtree rooted at $v$, including $v$ itself. Then we look at all nodes $v$ for which $N(v)$ is at least $k$. Among those, pick a node $v$ for which $N(v)$ is minimum. Then $N(v)$ is at least $k$ and each of its children has a subtree of size at most $k - 1$. Now we make $B_j$: Take a leaf in the subtree of $v$, add it to $B_j$, and remove it from the tree. Repeat this until $B_j$ has size $k$.

With the two partitions $A_1, \ldots, A_n$ and $B_1, \ldots, B_n$ in hand, we color the points of $P$ by $k$ colors $c_1, \ldots, c_k$ as in Theorem 2. Thus in each $A_i$ and in each $B_i$ we get $k$ distinct colors.

Notice that between any two points in the subtree of $v$ there is a path in $T$ with at most $2k - 2$ edges. We say that $v$ is the representative of $B_j$. Moreover, we define the parent of $B_j$ to be the bucket containing $v$ (if $v \notin B_j$) or the bucket containing $v$’s parent (if $v \in B_j$). For each color $c_i$ we construct $T_i$ as follows: for each bucket $B_j$ we connect its point with color $c_i$ (say point $r_i$) to the point with color $c_i$ in $B_j$’s parent bucket (say point $p_i$). To prove the approximation ratio it suffices to show that between $r_i$ and $p_i$ there is a path of length at most $3k - 2$ in $T$. This is easily seen as there is a path of length at most $k - 1$ from $r_i$ to the representative of $B_j$, say $v$, and there is an edge from $v$ to a node $u$ in $B_j$’s parent bucket, and there is a path of length at most $2k - 2$ between $u$ and $p_i$ in the parent bucket. The following theorem summarizes our result.
Theorem 3. There exists a polynomial-time \((3k - 2)\)-approximation algorithm for the \(k\)-disjoint bottleneck spanning tree problem on points in a metric space.

Remark. The length \(2k - 2\) within each bucket of size \(k\) is the best achievable. For example consider a tree rooted at a node \(v\) with \(k + 1\) subtrees each is a path with \(k - 1\) nodes. This tree has \(k^2\) nodes in total which will be partitioned into \(k\) buckets of size \(k\). Since there are \(k + 1\) leaves at least two of them lie in the same bucket (by the pigeonhole principle), and thus the distance between them will be \(2k - 2\).

3 The 2-GBST problem

In this section we study the 2-GBST problem: Given a set \(P\) of \(n\) points in some metric space that is partitioned into clusters of size at most 2, find a tree that contains exactly one point from each cluster and minimizes the largest edge length. First we prove the hardness of this problem and then present an approximation algorithm.

Theorem 4. Unless \(P = NP\), there is no polynomial-time algorithm that approximates the metric 2-generalized bottleneck spanning tree problem by a factor better than 2.

Proof. We use a reduction form the traditional 3-SAT problem: given a boolean expression \(E\) as the conjunction of clauses, each of which is the disjunction of three distinct literals (a variable or its negation), decide whether \(E\) is satisfiable.

Given any instance of the 3-SAT problem consisting of an expression \(E\) with \(m\) clauses \(C_1, \ldots, C_m\) and \(n\) variables \(x_1, \ldots, x_n\) we construct an instance of the 2-GBST problem consisting of a metric graph \(G\) as follows (the vertices of \(G\) represent points in a metric space). For each clause \(C_j\) create a cluster with one vertex \(p_j\). For each variable \(x_i\) create a cluster with two literal vertices \(v_i\) and \(\overline{v_i}\) that correspond to positive literal \(x_i\) and negative literal \(\overline{x_i}\), respectively. Create a cluster with one vertex \(r\). To simplify our description we use vertices and their corresponding clauses or literals interchangeably. Connect each literal vertex, by edges of weight 1, to vertices \(p_j\) of all clauses \(C_j\) that they appear in. Connect \(r\) to all literal vertices by edges of weight 1. All other edges of \(G\) have weight 2. Notice that \(G\) is a metric graph with \(m + 2n + 1\) vertices. We show that \(E\) is satisfiable if and only if \(G\) has a generalized spanning tree with edges of weight 1. This would imply the statement of the theorem because (by contraposition) any approximation algorithm with factor less than 2 would give a tree with edges of weight 1, and thus could solve the 3-SAT problem.

First suppose that \(E\) is satisfiable, and consider a truth assignment of variables that satisfies \(E\). We obtain a tree \(T\) as follows. For the vertex set of \(T\) we select \(r\), all vertices \(p_j\), and each \(v_i\) (if \(x_i\) is true) or \(\overline{v_i}\) (if \(x_i\) is false). For the edge set of \(T\) we connect \(r\) to every selected literal vertex, and we connect each \(p_j\) to exactly one selected literal vertex that satisfies \(C_j\). The tree \(T\) is a feasible solution for the 2-GBST problem on \(G\) (as it contains exactly one vertex from each cluster) and all its edges have weight 1.

For the other direction assume that \(T\) is a generalized spanning tree of \(G\) with edges of weight 1. The tree \(T\) should contain \(r\) and all vertices \(p_j\) because they are the only vertices in their clusters. For each \(p_j\) only edges of \(G\) that connect \(p_j\) to literal vertices have weight 1. Thus each \(p_j\) is connected to at least one literal vertex in \(T\). Moreover \(T\) contains exactly one vertex from each
cluster \( \{v_i, \overline{v_i}\} \) of literal vertices. Therefore, by setting \( x_i \) as true (if \( T \) contains \( v_i \)) or false (if \( T \) contains \( \overline{v_i} \)) we obtain a satisfying assignment for \( E \). \( \square \)

If in the proof of Theorem 4 we replace all edge-weights of 2 with an arbitrary large constant, we obtain the following corollary.

**Corollary 1.** It is NP-hard to approximate the non-metric 2-generalized bottleneck spanning tree problem within any constant factor.

If we were interested in generalized minimum spanning trees, then our reduction in the proof of Theorem 4 would also give a short proof for the NP-hardness of the metric 2-GMST problem: It can be verified that \( E \) is satisfiable if and only if \( G \) has a generalized spanning tree of total weight \( m + n \). We note the existence of (somewhat involved) proofs for the hardness of the Euclidean 2-GMST problem; see the thesis of Fraser [16, page 140] (reduction from maximum 2-SAT), the paper of Ataei et al. [22] (reduction from planar 3-SAT), and a recent result of Dey et al. [13] (reduction from maximum 2-SAT).

### 3.1 A 3-approximation for the 2-GBST

Here we present our 3-approximation algorithm for the 2-GBST problem on a set \( P \) of \( n \) points in a metric space that is partitioned into \( m \) clusters \( C_1, \ldots, C_m \), each of size at most 2. Notice that \( n/2 \leq m \leq n \). Let \( \lambda^* \) be the bottleneck of a fixed optimal solution. In a nutshell, our algorithm works as follows. First we compute a tree \( T_1 \) that contains “at least” one point from each cluster and its bottleneck is at most \( \lambda^* \). Then we obtain a tree \( T_2 \) from \( T_1 \) that contains “exactly” one point from each cluster and its bottleneck is at most thrice \( \lambda^* \). Therefore

\[
\lambda(T_2) \leq 3 \cdot \lambda(T_1) \leq 3 \cdot \lambda^*,
\]

which means that \( T_2 \) is a 3-approximate solution for the 2-GBST problem. In the rest of this section we show how to construct \( T_1 \) and \( T_2 \). Our algorithm for computing \( T_2 \) from \( T_1 \) is of independent interest. The running time of our algorithm is dominated by the computation of a minimum spanning tree. The following theorem summarizes our result.

**Theorem 5.** There exists a polynomial-time 3-approximation algorithm for the 2-generalized bottleneck spanning tree problem on points in a metric space.

### 3.1.1 Construction of \( T_1 \)

First we make an empty graph \( G \) over the \( n \) points of \( P \). Then we add edges between the points of \( G \) in a non-decreasing order of the distances, and stop as soon as \( G \) has a connected component, say \( C \), that contains at least one point from each cluster. All edges of \( C \) are of length at most \( \lambda^* \). Now we compute \( T_1 \) as an arbitrary spanning tree of \( C \).

**Remark.** When the running time is a concern, one can guess \( \lambda^* \) in a binary search fashion to speed up the algorithm. Also, it is possible to compute \( T_1 \) as a subtree of the minimum spanning tree of \( P \). In this case, the total running time is dominated by the computation of the minimum spanning tree; the details are removed as we are not concerned about the running time here.
3.1.2 Construction of $T_2$

In this section we prove the following theorem.

**Theorem 6.** Given a tree $T_1$ and a partitioning of its nodes into clusters of size at most two, we can obtain a tree $T_2$ that contains exactly one node from each cluster and the length of its edges is at most 3 in the metric of $T_1$. The upper bound 3 is the best achievable.

First we show that the distance 3 (in the metric of $T_1$) is the best achievable upper bound. Figure 3 illustrates a tree $T_1$ as a path with eight nodes. The nodes of $T_1$ are partitioned into five clusters $\{a\}, \{b_1, b_2\}, \{c_1, c_2\}, \{d_1, d_2\}, \{e\}$. To obtain $T_2$ we have to choose points $a$ and $e$ because they are the only points in their clusters. Due to symmetry we may choose $b_1$ from cluster $\{b_1, b_2\}$. In this case if we do not choose $d_2$ then the distance of $e$ to its closest point in $T_2$ would be at least 3, thus we may assume $d_2$ is chosen. In this setting, if we choose $c_1$ (as depicted in Figure 3) then the distance between $c_1$ and $d_2$ will be 3, and if we choose $c_2$ then the distance between $b_1$ and $c_2$ will be 3. Thus, in all cases we get an edge of length 3 in $T_2$.

![Figure 3: Illustration of the lower bound 3. Dashed lines represent two nodes belonging to the same cluster. The black vertices are chosen for $T_2$.](image)
claim (proved below) that for each selected node $a$ at any level of $T_1$ (except for the root) there exists a selected node $b$ at a higher level such that the path between $a$ and $b$ in $T_1$ has at most three edges, i.e. the distance between $a$ and $b$ is at most 3 in the metric of $T_1$. For each selected node $a$, we add the edge $(a,b)$ to $T_2$. As each $a$ is connected to a node in a higher level, all nodes of $T_2$ are connected (via root) and hence it is a tree.

Figure 4: (a) Node selection (dashed lines represent two nodes in the same cluster): red squares (belong to clusters of size one) and black nodes (belong to clusters of size two) are selected whereas the white nodes (paired with black nodes) are burned. (b) Illustration for the edge length of $T_2$.

Now we verify the above claim. Let $a_1$ be the parent of $a$, as in Figure 4(b). If $a_1$ is selected then set $b = a_1$ and we are done. Assume that $a_1$ is burned. Let $a_2$ be the parent of $a_1$. If $a_2$ is selected then set $b = a_2$ and we are done. Assume that $a_2$ is also burned. Notice that $a_2$ was burned before $a_1$ was, because otherwise the selection process would select $a_2$ right after burning $a_1$. Right after burning $a_2$ the process have checked the parent of $a_2$ which we denote by $a_3$. If $a_3$ was open then it would have been selected, and thus we set $b = a_3$ and we are done. If $a_3$ was burned then the process would have checked the children of $a_2$ and have selected a child $a'_1$ because $a_2$ had an open child which was $a_1$; this case is depicted in Figure 4(b). In this case we set $b = a'_1$ and we are done. The existence of $a_1$, $a_2$, and $a_3$ comes from the fact that the root of $T_1$ is a selected node.

Remark. It might be tempting to use our 3-approximation algorithm for the 2-GBST problem to obtain a 3-approximation for the 2-DBST problem, say by coloring the selected nodes red and the burned nodes blue. This may not be an easy task because each time the process starts by selecting an arbitrary open node, these selected nodes could form a long path between burned nodes.

4 The $k$-PBST problem

Let $k \geq 2$ be an integer. In this section we study the $k$-PBST problem: Given $kn$ points in some metric space, find $k$ trees each containing exactly $n$ points and minimize the largest edge length (over all edges of the $k$ trees). First we prove the hardness of this problem. We assume that $n$ is at least 3, because if $n = 2$ then the problem is equivalent to the bottleneck matching problem which can be solved in polynomial time. Then we present an approximation algorithm for this problem.

Theorem 7. Unless $P = NP$, there is no polynomial-time algorithm that approximates the metric $k$-partition bottleneck spanning tree problem by a factor better than 2, for any $k \geq 2$. 

11
Theorem 8. There exists a polynomial-time desirable we present algorithms that achieve
the upper bounds 2 and 3 for the edge lengths are the best achievable.

Given a tree $T$, we start by computing a minimum spanning tree $e$ of all points. Let $T'$ and $T''$ be the two trees obtained by removing $e$ from $T$. If the number of nodes in $T'$ and in $T''$ are multiples of $n$, say $i \cdot n$ and $j \cdot n$ where $i + j = k$, then we recursively construct $i$ trees on the nodes of $T'$ and $j$ trees on the nodes of $T''$.

Assume that the number of nodes in $T'$ and $T''$ are not multiples of $k$. Then the optimal solution must have an edge between a node of $T'$ and a node of $T''$. The length of this edge is at least $w(e)$, and thus $\lambda^* \geq \lambda(T)$. Then by Theorem 9 we obtain $k$ trees on the nodes of $T$ such that their edge lengths are at most $\alpha(k) \cdot \lambda^*$, where $\lambda^*$ is the bottleneck of a fixed optimal solution.

We use a reduction from the NP-hard problem of partitioning the vertex set of a graph $G = (V, E)$ into $k$ ($2 \leq k \leq |V|/3$) equal-size subsets $V_1, \ldots, V_k$ such that the induced subgraph by each $V_i$ is connected [15]. Let $G'$ be the complete edge-weighted graph obtained by adding edges to $G$ and then assigning weight 1 to every edge of $E$ and weight 2 to every other edge. Notice that $G'$ is a metric graph with $|V|$ vertices. It is easily seen that the partition problem on $G$ has a solution if and only if $G'$ contains $k$ equal-size spanning trees with edges of weight 1. The inapproximability claim follows because any approximation algorithm with factor less than 2 would give spanning trees with edges of weight 1, which would solve the partitioning problem on $G$.

If in the proof of Theorem 7 we replace all edge-weights of 2 with an arbitrary large constant, we obtain the following corollary.

Corollary 2. It is NP-hard to approximate the non-metric $k$-partition bottleneck spanning tree problem within any constant factor, for any $k \geq 2$.

4.1 Approximating the $k$-PBST

Now we present an $\alpha(k)$-approximation algorithm for the $k$-PBST problem, where $\alpha(k) = 2$ for $k = 2, 3$ and $\alpha(k) = 3$ for $k \geq 4$. In view of Theorem 7 the factor 2 is the best achievable for $k = 2, 3$. Given $kn$ points in a metric space, we show how to construct $k$ trees each containing exactly $n$ points and their largest edge length is at most $\alpha(k) \cdot \lambda^*$, where $\lambda^*$ is the bottleneck of a fixed optimal solution.

We start by computing a minimum spanning tree $T$ of all points. Let $e$ be a longest edge of $T$, that is $\lambda(T) = w(e)$. Let $T'$ and $T''$ be the two trees obtained by removing $e$ from $T$. If the number of nodes in $T'$ and in $T''$ are multiples of $n$, say $i \cdot n$ and $j \cdot n$ where $i + j = k$, then we recursively construct $i$ trees on the nodes of $T'$ and $j$ trees on the nodes of $T''$.

Theorem 8. There exists a polynomial-time $\alpha$-approximation algorithm for the $k$-partition bottleneck spanning tree problem on points in a metric space where $\alpha = 2$ for $k = 2, 3$ and $\alpha = 3$ for $k \geq 4$. The approximation factor 2 for $k = 2, 3$ is the best achievable in polynomial time.

4.2 Balanced tree partitioning theorem

In this section we prove the following theorem. We denote the number of nodes of a tree $T$ by $|T|$.

Theorem 9. Given a tree $T$ with $kn$ nodes we can obtain $k$ disjoint trees $T_1, \ldots, T_k$ each containing exactly $n$ nodes of $T$ such that

1. If $k = 2, 3$ then the length of edges in each $T_i$ is at most 2 in the metric of $T$.
2. If $k \geq 4$ then the length of edges in each $T_i$ is at most 3 in the metric of $T$.

The upper bounds 2 and 3 for the edge lengths are the best achievable.

For the proof we first show that the upper bounds 2 and 3 are the best achievable. Then we present algorithms that achieve desirable trees $T_1, \ldots, T_k$ with the claimed edge lengths. The lengths mentioned in our proof are in the metric of $T$.  

Upper bounds. It is easily seen that the upper bound of 2 is the best achievable (for \( k = 2, 3 \)) for example when \( T \) is a star with 3 and 5 leaves, respectively.

To verify that 3 is the best achievable upper bound (for \( k \geq 4 \)) consider a tree \( T \) rooted at a node \( v \) with \( k + 1 \) subtrees each is a path with \( k - 1 \) nodes; see the figure to the right for \( k = 4 \). The tree \( T \) has \( k^2 \) nodes. Let \( R \) be the set of \( k + 1 \) nodes that are at distance 1 from \( v \). Each node of \( R \) represents a path connected to \( v \). Now consider any set of \( k \) disjoint trees \( T_1, \ldots, T_k \) each consisting of \( k \) nodes of \( T \). We show, by contradiction, that the length of an edge in some \( T_i \) is at least 3. After a suitable relabeling assume that \( v \) belongs to \( T_1 \). Then each tree \( T_i \) with \( i \in \{2, \ldots, k\} \) should have nodes from at least two of the paths connected to \( v \) because each path itself has \( k - 1 \) nodes. In particular \( T_i \) should contain the representatives of these paths because otherwise \( T_i \) should have an edge of length at least 3. Thus each \( T_i \) contains at least two distinct nodes from \( R \). This implies that \( |R| \geq 2(k - 1) \). Combining this inequality with the fact that \( |R| = k + 1 \), implies that \( k \leq 3 \) which is a contradiction.

Algorithm for \( k \geq 4 \). Our algorithm for \( k \geq 4 \) uses the fact that the cube of \( T \) is Hamiltonian. It is implied from a result of Karaganis [23] and independently from a result of Lesniak [25] that in polynomial time we can find a Hamiltonian path on nodes of \( T \) with edges of length at most 3. By cutting this path into \( k \) equal-size pieces we obtain \( k \) desired trees.

Remark. One could simply obtain a 2-approximation if the square\(^3\) of \( T \) has a Hamiltonian path. However, this property holds only for a very restricted class of trees called horsetail [37].

Algorithm for \( k = 2 \). We show how to find two disjoint trees \( R \) and \( B \) each containing exactly \( n \) nodes of \( T \) and the length of their edges is at most 2. To simplify our description we assume that the nodes of \( R \) and \( B \) are colored red and blue, respectively.

We root \( T \) at a leaf \( r \), as in Figure 5. Then \( r \) has only one child which we denote by \( b \). Assume that \( r \) is at level 1, its child \( b \) is at level 2, the children of \( b \) are at level 3, and so on. Color all nodes at odd levels red and color all nodes at even levels blue. Compute a rooted tree \( R \) on red points by connecting each red node to its grandparent, and compute a rooted tree \( B \) on blue points by connecting each blue node to its grandparent, as in Figure 5. Notice that \( R \) is rooted at \( r \) and \( B \)

\[^3\]The square of a graph \( G \) has the same vertices as \( G \), and has an edge between two distinct vertices if and only if there exists a path, with at most two edges, between them in \( G \).
is rooted at $b$. Since each red node (resp. blue node) is connected to its grandparent, every edge of $R$ (resp. $B$) has length 2.

If $R$ has $n$ nodes, so does $B$, and hence $\{R, B\}$ is a 2-approximate solution. If one tree, say $R$, has more than $n$ nodes, then we iteratively remove a leaf from $R$ until it is left with exactly $n$ nodes. We color the removed nodes of $R$ by blue; see Figure 5-right. Then we recompute the tree $B$ from the beginning by connecting each blue node to one of its parent and grandparent that is blue. Since no new edge is introduced in $R$, its edges still have length 2. Since each blue node is connected to its parent or grandparent (in $T$), the length of its edges is at most 2. Therefore, the new trees $R$ and $B$ are desirable.

Remark. It is easily seen that the above algorithm can be extended to obtain trees $R$ and $B$ of different sizes (as long as $\lvert R \rvert + \lvert B \rvert = \lvert T \rvert$) with the same upper bound of 2 on their edge lengths.

Algorithm for $k = 3$. Notice that $T$ has $3n$ nodes. We show how to find three disjoint trees $R$, $G$, and $B$ each containing exactly $n$ nodes of $T$ and the length of their edges is at most 2.

We root $T$ at a leaf node. For each node $v$ in $T$, let $N(v)$ denote the number of nodes in the subtree rooted at $v$; the node $v$ is counted. Then we look at all nodes $v$ for which $N(v)$ is at least $n$. Among those, pick a node $v$ for which $N(v)$ is minimum. Then $N(v)$ is at least $n$ and each of its children has a subtree of size at most $n - 1$. Observe that $v$ is not the root.

If $N(v) = n$ then we take the subtree rooted at $v$ as $R$, remove $R$ from $T$, and then obtain two trees $G$ and $B$ from the new tree $T$ (which now has $2n$ nodes) using our algorithm for $k = 2$.

Assume that $N(v) > n$. Then $v$ has at least two children which we denote by $u_1, u_2, \ldots, u_m$ where $m \geq 2$. Let $U_j$ denote the subtree rooted at $u_j$. Take the smallest index $j$ in $\{1, \ldots, m\}$ for which $|U_1| + \cdots + |U_j| \geq n$. Then $|U_1| + \cdots + |U_{j-1}| < n$. Let $n_1 = n - (|U_1| + \cdots + |U_{j-1}|)$. Let $U_j^v$ be the subtree consisting of $U_j$ and the node $v$ together with the edge connecting $v$ to $u_j$. We use our algorithm for $k = 2$ to obtain from $U_j^v$ two trees $T_j^v$ and $T_j''$ of sizes $n_1'$ and $|U_j''| - n_1' = |U_j| + 1 - n_1'$, respectively, such that $T_j^v$ is rooted at $u_j$, $T_j''$ is rooted at $v$, and their edge lengths are at most 2; see Figure 6. Now we obtain $R$ by taking the trees $U_1, \ldots, U_{j-1}$, and $T_j^v$ and interconnecting their roots to form one tree. Notice that $R$ has $n$ nodes and its edge lengths are at most 2. We remove the nodes of $R$ from $T$. We also remove all edges of $T$ that lie in $U_j$, and add the edges of $T_j''$ (which are of length at most 2) to $T$. Notice that $\lvert T_j'' \rvert < n$ because it does not have $u_j$ (although it contains $v$). To obtain $G$ and $B$ we consider the following cases depending on the number $N(v)$ in the new tree $T$ which has $2n$ nodes:

- $N(v) = n$. In this case we take the subtree rooted at $v$ as $G$, remove $G$ from $T$, and then take the resulting tree $T$ (which now has $n$ nodes) as $B$.

- $N(v) < n$. We walk up the tree $T$ from $v$ and stop at the first node $w$ for which $N(w) \geq n$. We repeat the above process to obtain $G$ (which is now playing the role of $R$) but we denote the subtree of $w$ that contains $v$ by $U_1$. This ensures that the edges of $T_j''$ will appear in $G$ without getting longer. After obtaining $G$, the remaining part of $T$ will form the tree $B$.

- $N(v) > n$. See Figure 6. In this case we somehow repeat a procedure similar to what we did to obtain $R$. Let $l \in \{j + 1, \ldots, m\}$ be the smallest index for which $\lvert T_j^v \rvert + |U_{j+1}| + \cdots + |U_l| \geq n$. Notice that $U_j$ exists because $N(v) > n$. Then $\lvert T_j^v \rvert + |U_{j+1}| + \cdots + |U_l| < n$. Let $n_2 = n - (|T_j^v| + |U_{j+1}| + \cdots + |U_{l-1}|) + 1$ (the addition of 1 will become clear shortly). Let $U^v_j$ be the subtree consisting of $U_l$ and the node $v$ together with the edge connecting $v$ to $u_l$ (notice that $v$ also belongs to $T_j^v$). We use our algorithm for $k = 2$ one more time to obtain
from $U_i^v$ two trees $T'_i$ and $T''_i$ of sizes $n'_2$ and $|U_i^v| - n'_2 = |U_i| + 1 - n'_2$, respectively, such that $T'_i$ is rooted at $v$, $T''_i$ is rooted at $u_i$ and their edge lengths are at most 2. Now we obtain $G$ by taking the trees $T''_j, U_{j+1}, \ldots, U_{l-1}$, and $T''_l$ and then interconnecting their roots to form one tree. The tree $G$ has $n$ nodes (without double counting $v$ which is in both $T''_j$ and $T''_l$) and its edge lengths are at most 2. We obtain the third tree, i.e. $B$, as follows. We remove the nodes of $G$ from $T$. By interconnecting the roots of $T''_l, U_{l+1}, \ldots, U_m$ together and then connecting $u_m$ to the parent of $v$ (which exists) we obtain the tree $B$.

**Remark.** To see why the above procedure cannot be extended to the case of $k = 4$, assume that $N(v) > n$ after the removal of $G$ from $T$. As $v$ is already used for making $G$ we cannot reuse it to make another tree, and hence we will be forced to introduce longer edges.

### 5 Conclusions

A natural open problem is to improve the presented approximation ratios further. Most of our approximation ratios consider the largest edge length of the standard BST as the lower bound. A better lower bound for the largest edge length of an optimal solution (not the standard BST) would improve the approximation ratios. It would be interesting to explore whether our algorithm for the 2-GBST problem could be extended to an $O(k)$-approximation algorithm for the $k$-GBST problem. Also it would be interesting to verify whether the approximation ratio of 3 for the $k$-PBST problem ($k \geq 4$) is tight, knowing that 2 is a lower bound.

### References

[1] M. Andersson, J. Gudmundsson, C. Levcopoulos, and G. Narasimhan. Balanced partition of minimum spanning trees. *International Journal of Computational Geometry & Applications*, 13(4):303–316, 2003.
[2] E. M. Arkin, A. Banik, P. Carmi, G. Citovsky, S. Jia, M. J. Katz, T. Mayer, and J. S. B. Mitchell. Network optimization on partitioned pairs of points. In Proceedings of the 28th International Symposium on Algorithms and Computation (ISAAC), pages 6:1–6:12, 2017.

[3] E. M. Arkin and R. Hassin. Minimum-diameter covering problems. Networks, 36(3):147–155, 2000.

[4] I. Averbakh and O. Berman. A heuristic with worst-case analysis for minimax routing of two travelling salesmen on a tree. Discrete Applied Mathematics, 68(1-2):17–32, 1996.

[5] I. Averbakh and O. Berman. \((p - 1)/(p + 1)\)-approximate algorithms for \(p\)-traveling salesmen problems on a tree with minmax objective. Discrete Applied Mathematics, 75(3):201–216, 1997.

[6] B. K. Bhattacharya, A. Custic, A. Rafiey, A. Rafiey, and V. Sokol. Approximation algorithms for generalized MST and TSP in grid clusters. In Proceedings of the 9th International Conference on Combinatorial Optimization and Applications (COCOA), pages 110–125, 2015.

[7] V. Blanco, E. Fernández, and J. Puerto. Minimum spanning trees with neighborhoods: Mathematical programming formulations and solution methods. European Journal of Operational Research, 262(3):863–878, 2017.

[8] J. A. Bondy and U. S. R. Murty. Graph Theory with Applications. Macmillan Education UK, 1976.

[9] P. M. Camerini. The min-max spanning tree problem and some extensions. Information Processing Letters, 7(1):10–14, 1978.

[10] Y. Chen, Z. Chen, G. Lin, Y. Xu, and A. Zhang. Approximation algorithms for maximally balanced connected graph partition. In Proceedings of the 13th International Conference on Combinatorial Optimization and Applications (COCOA), pages 130–141, 2019.

[11] J. Chlebíková. Approximating the maximally balanced connected partition problem in graphs. Information Processing Letters, 60(5):223–230, 1996.

[12] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. Introduction to Algorithms. McGraw-Hill, Cambridge, 1989.

[13] S. Dey, R. K. Jallu, and S. C. Nandy. On the minimum spanning tree problem in imprecise set-up. arXiv:2104.04229, 2021.

[14] R. Dorrigiv, R. Fraser, M. He, S. Kamali, A. Kawamura, A. López-Ortiz, and D. Seco. On minimum- and maximum-weight minimum spanning trees with neighborhoods. Theory of Computing Systems, 56(1):220–250, 2015.

[15] M. E. Dyer and A. M. Frieze. On the complexity of partitioning graphs into connected subgraphs. Discrete Applied Mathematics, 10(2):139–153, 1985.

[16] R. Fraser. Algorithms for geometric covering and piercing problems. Phd thesis, University of Waterloo, 2012.

[17] H. N. Gabow, S. N. Maheswari, and L. J. Osterweil. On two problems in the generation of program test paths. IEEE Transactions on Software Engineering, 2(3):227–231, 1976.
[18] M. X. Goemans and D. Bertsimas. Survivable networks, linear programming relaxations and the parsimonious property. *Mathematical Programming*, 60:145–166, 1993.

[19] P. Hall. On representatives of subsets. *Journal of the London Mathematical Society*, 10(1):26–30, 1935.

[20] E. Halperin and R. Krauthgamer. Polylogarithmic inapproximability. In *Proceedings of the 35th Annual ACM Symposium on Theory of Computing (STOC)*, pages 585–594, 2003.

[21] M. P. Johnson. Red-blue-partitioned MST, TSP, and matching. In *Proceedings of the 30th Canadian Conference on Computational Geometry (CCC)G*, pages 259–264, 2018.

[22] H. A. Kachooei, M. Davoodi, and D. Tayebi. On the generalized minimum spanning tree in the Euclidean plane. In *Proceedings of the 1st Iranian Conference on Computational Geometry (ICCG)*, pages 19–23, 2018.

[23] J. Karaganis. On the cube of a graph. *Canadian Mathematical Bulletin*, 11(2):295–296, 1968.

[24] S. Karakawa, E. Morsy, and H. Nagamochi. Minmax tree cover in the euclidean space. *Journal of Graph Algorithms and Applications*, 15(3):345–371, 2011.

[25] L. Lesniak. Graphs with 1-Hamiltonian-connected cubes. *Journal of Combinatorial Theory, Series B*, 14(2):148–152, 1973.

[26] M. Löffler. *Data Imprecision in Computational Geometry*. Phd thesis, Utrecht University, 2009.

[27] J. S. B. Mitchell. A PTAS for TSP with neighborhoods among fat regions in the plane. In *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 11–18, 2007.

[28] J. S. B. Mitchell. A constant-factor approximation algorithm for TSP with pairwise-disjoint connected neighborhoods in the plane. In *Proceedings of the 26th ACM Symposium on Computational Geometry (SoCG)*, pages 183–191, 2010.

[29] Y. Myung, C. Lee, and D. Tcha. On the generalized minimum spanning tree problem. *Networks*, 26(4):231–241, 1995.

[30] H. Nagamochi and T. Kawada. Approximating the minmax subtree cover problem in a cactus. In *Proceedings of the 15th International Symposium on Algorithms and Computation (ISAAC)*, pages 705–716, 2004.

[31] H. Nagamochi and T. Kawada. Minmax subtree cover problem on cacti. *Discrete Applied Mathematics*, 154(8):1254–1263, 2006.

[32] H. Nagamochi and K. Okada. Polynomial time 2-approximation algorithms for the minmax subtree cover problem. In *Proceedings of the 14th International Symposium on Algorithms and Computation (ISAAC)*, pages 138–147, 2003.

[33] H. Nagamochi and K. Okada. A faster 2-approximation algorithm for the minmax $p$-traveling salesmen problem on a tree. *Discrete Applied Mathematics*, 140(1-3):103–114, 2004.
[34] P. C. Pop. The generalized minimum spanning tree problem: An overview of formulations, solution procedures and latest advances. *European Journal of Operational Research*, 283(1):1–15, 2020.

[35] P. C. Pop, W. Kern, G. Still, and U. Faigle. Relaxation methods for the generalized minimum spanning tree problem. *Electronic Notes in Discrete Mathematics*, 8:76–79, 2001.

[36] P. C. Pop, G. Still, and W. Kern. An approximation algorithm for the generalized minimum spanning tree problem with bounded cluster size. In *Proceedings of the 1st Workshop on Algorithms and Complexity in Durham*, pages 115–121, 2005.

[37] J. Radoszewski and W. Rytter. Hamiltonian paths in the square of a tree. In *Proceedings of the 22nd International Symposium on Algorithms and Computation (ISAAC)*, pages 90–99, 2011.

[38] Y. Wakabayashi, F. Chataigner, and L. B. Salgado. Approximation and inapproximability results on balanced connected partitions of graphs. *Discrete Mathematics & Theoretical Computer Science*, 9(1), 2007.