A Constant Factor Approximation for Capacitated Min-Max Tree Cover

Syamantak Das
IIIT Delhi, India
http://faculty.iiitd.ac.in/~syamantak/
syamantak@iiitd.ac.in

Lavina Jain
IIIT Delhi, India
Google scholar page
lavina16052@iiitd.ac.in

Nikhil Kumar
IIT Delhi, India
Google scholar page
nikhil@cse.iitd.ac.in

Abstract

Given a graph $G = (V, E)$ with non-negative real edge lengths and an integer parameter k, the (uncapacitated) Min-Max Tree Cover problem seeks to find a set of at most k trees which together span V and each tree is a subgraph of G. The objective is to minimize the maximum length among all the trees. In this paper, we consider a capacitated generalization of the above and give the first constant factor approximation algorithm. In the capacitated version, there is a hard uniform capacity (λ) on the number of vertices a tree can cover. Our result extends to the rooted version of the problem, where we are given a set of k root vertices, R and each of the covering trees is required to include a distinct vertex in R as the root. Prior to our work, the only result known was a $(2k-1)$-approximation algorithm for the special case when the total number of vertices in the graph is $k\lambda$ [Guttmann-Beck and Hassin, *J. of Algorithms, 1997*]. Our technique circumvents the difficulty of using the minimum spanning tree of the graph as a lower bound, which is standard for the uncapacitated version of the problem [Even et al.,*OR Letters 2004*] [Khani et al.,*Algorithmica 2010*]. Instead, we use *Steiner trees* that cover λ vertices along with an iterative refinement procedure that ensures that the output trees have low cost and the vertices are well distributed among the trees.

2012 ACM Subject Classification Theory of computation → Routing and network design problems

Keywords and phrases Approximation Algorithms, Graph Algorithms, Min-Max Tree Cover, Vehicle Routing, Steiner Tree

Digital Object Identifier 10.4230/LIPIcs.APPROX/RANDOM.2020.55

Category APPROX

1 Introduction

Covering vertices of a given graph using simpler structures, for example, paths, trees, stars and so on, have long attracted the attention of the Computer Science and Operations Research communities. This can be attributed to a variety of applications in vehicle routing, network design and related problems. One classical example is the so-called "Nurse Location Problem" [10]. The goal is to place a group of nurses at different locations and finding a tour for each of them so that every patient is visited by a nurse. A similar setting arises in vehicle routing. Suppose we are given a set of vehicles, initially located at a given set of depots. The goal is to find a tour for each of these vehicles, each starting and ending at the respective depots so as to cover client demands at various locations. One of the most popular objectives is to minimize the maximum distance travelled by any vehicle, also known as the

Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM 2020).
Editors: Jarosław Byrka and Raghu Meka; Article No. 55; pp. 55:1–55:13

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

makespan of the solution. A standard reduction shows that this problem is equivalent, within an approximation factor of 2, to finding trees in a graph such that all vertices in the graph are covered by the union of these trees. In this paper, we consider a natural generalization of the above setting. As before, we are given a set of vehicles, initially located at a given set of depots and a set of clients. Additionally one package is to be delivered to each of the clients and each vehicle can carry at most a fixed number of packages (all packages are identical). We have to find a set of tour such that each client receives a package and the objective, as before is to minimize the maximum distance travelled by any vehicle. This can be seen as a capacitated version of the Nurse Location Problem, where each nurse can visit at most a given number of patients. Within a factor of two, this problem is equivalent to the following generalization of the TSP problem : given a metric and two constants k, λ, find k tours, each tour on no more than λ vertices, such that the maximum cost over all the tours is minimized. We formally define the problem now, which will be useful in further discussions.

1.1 Notations and Preliminaries

We set up some preliminaries, notations and definitions from literature that will be useful in the exposition of our contributions. The set of positive integers $\{1, 2, , \ldots, n\}$ is denoted by $[n]$. Given a graph $G = (V, E)$, $H = (V_H, E_H)$ is a subgraph of G if $V_H \subseteq V$ and $E_H \subseteq E$. We use $\ell(H)$ to denote the total length of edges in the subgraph H. For any two vertices u, v, $d(u, v)$ denotes the shortest path distance between u, v. We extend the definition to subsets of vertices U, V - define $d(U, V)$ to be $\min_{u \in U, v \in V} d(u, v)$. The set of vertices in any graph H is denoted by $V(H)$.

▶ **Definition 1** (Min-Max Tree Cover Problem (MMTC)). *Given a graph $G(V, E)$, edge length $\ell : E \to \mathbb{R}_{\geq 0}$ and a parameter k, one is required to output a set of k trees T_i for $i \in [k]$, such that each T_i is a subgraph of G and $\cup_{i=1}^{k} V(T_i) = V$. The objective is to minimize $max_{i=1}^{k} \ell(T_i)$.*

Note that two trees in a feasible solution can share vertices as well as edges. In the capacitated version of the problem, we are also given an additional parameter λ. A feasible solution to the Capacitated Min-Max k-Tree Cover consists of a set of trees (not necessarily disjoint) $\mathcal{T} = \{T_1, T_2, \ldots, T_k\}$ along with an assignment of each vertex $v \in V$ to one of the trees containing it, such that no more than λ vertices are assigned to any $T \in \mathcal{T}$. The goal, as in the case of uncapacitated case, is to minimize the maximum length of any tree in $\mathcal{T}$. Let $\text{cov}(T)$ denote the set of vertices assigned to T. We think of the vertices in $\text{cov}(T)$ as being covered by T. Hence, in any feasible assignment $\cup_{T \in \mathcal{T}} \text{cov}(T) = V$ and $|\text{cov}(T)| \leq \lambda$ for all $T \in \mathcal{T}$. Note that a vertex may be a part of multiple trees, but it is covered by exactly one of them. Whenever the notation $\text{cov}(T)$ is used, there is an (implicit) underlying assignment of vertices to the trees. We do not explicitly mention the assignment if it is clear from the context.

▶ **Definition 2** (Capacitated Min-Max Tree Cover (CapMMTC)). *Given a graph $G(V, E)$, edge length $\ell : E \to \mathbb{R}_{\geq 0}$, and two integer parameters k, λ, one is required to output a set of k trees T_i for $i \in [k]$ along with an assignment of every vertex in G to a tree containing it, such that each T_i is a subgraph of G, $\cup_{i=1}^{k} \text{cov}(T_i) = V$ and $|\text{cov}(T_i)| \leq \lambda$ for $i \in [k]$. The objective is to minimize $max_{i=1}^{k} \ell(T_i)$.*

In rooted version of MMTC and CapMMTC, we are given a set of k roots as well. The only additional constraint being that each tree in the output should contain a distinct root. We will refer to these problems as Rooted Min-Max Tree Cover Problem(RMMTC) and Capacitated Rooted Min-Max Tree Cover (CapRMMTC).

1.2 Our Contribution

We give the first polynomial time constant factor approximation algorithm for CapMMTC and CapRMMTC.

▶ **Theorem 3.** *There is a polynomial time $\mathcal{O}(1)$-approximation algorithm for* CapMMTC.

▶ **Theorem 4.** *There is a polynomial time $\mathcal{O}(1)$-approximation algorithm for* CapRMMTC.

Our algorithms are much more intricate and involved than those which give an $\mathcal{O}(1)$ approximation for the uncapacitated case, as is generally the case with capaciated versions of many problems in combinatorial optimization, for instance k-median and facility location problems. These are the first approximation algorithms for both the problems, to the best of our knowledge. The only known result is a $(2k-1)$-approximation for the special case when the total number of vertices in the graph is $k\lambda$ and hence every tree must contain exactly λ vertices [12]. All our algorithms are combinatorial. We first show the result for the unrooted case, ie. CapMMTC and then extend the ideas to prove the result for the rooted problem CapRMMTC. We show that value of the constant in Theorem 3 and 4 is ≤ 300. For simplicity of exposition, we have not tried to optimize the constant. We believe that it should be possible to do so with some more effort and leave it as an open problem. We prove Theorem 3 in Section 2 and defer the proof of Theorem 4 to the Appendix, Section 3.

1.3 Related Work

Even et al. [10] and Arkin et al. [3] gave 4-approximation algorithms for both rooted and unrooted (uncapacitated) MMTC. Khani et al. improved the unrooted version to a 3-approximation [16], whereas Nagamochi and Okada [17] gave a $3-2/(k+1)$-approximation for the special case of the rooted version where all roots are co-located. On the other hand, the MMTC problem has been shown to be hard to approximate to a factor better than $3/2$, assuming $P \neq NP$ [20]. The problem has been considered when the underlying graph has special structure. The rooted version of the MMTC problem on a tree admits a PTAS, as shown recently by Becker and Paul [5], while the unrooted version has a $(2+\epsilon)$-approximation, given by Nagamochi and Okada [17]. Further, Chen and Marx gave fixed parameter tractable algorithms for the problem on a tree [8]. MMTC on a star is equivalent to the classical makespan minimization problem on identical machine, for which elegant EPTAS-es are well known [1]. In stark contrast, no approximation algorithms have been reported so far for the capacitated versions of these problems, to the best of our knowledge. Guttmann-Beck and Hassin gave an $(2k-1)$-approximation for the special case where the number of vertices in each the tree is exactly λ [12]. On the other special case of a star metric, an EPTAS result follows from the work on identical machine scheduling with capacity constraints [7]. Interestingly, the problem becomes inapproximable if one disallows sharing of vertices and edges between the trees in the solution [12], even when $k = 2$. A related but quite different setting is that of bounded capacity vehicle routing with the objective of minimizing the total length of all the tours. Here, a vehicle is allowed to make multiple tours to cover all the points, however, each tour can serve at the most λ clients. This problem has been well studied, a $2.5 - \lambda^{-1}$-approximation is known for general graphs [13], while Becker et al. have reported a PTAS on planar graphs recently [4]. Capacitated versions of other classical combinatorial optimization problems are very well studied and we give a highly non-exhaustive list here. Capacitated unweighted vertex cover admits a 2-approximation [14, 19]. Capacitated versions of clustering problems like k-center [2], k-median [6] and scheduling problems [18] have also been widely popular.

1.4 High Level Ideas and Techniques

Most algorithms for either MMTC or rooted MMTC build upon the following idea. Assume that we know the optimal tree cost - say $T^\star$. Further, for ease of exposition, let us assume that the graph is connected and does not contain any edge of length more than $T^\star$. Now consider the k trees in the optimal solution, each of total length at most $T^\star$. Adding at most $k-1$ edges to the union of these trees forms a spanning tree of the entire graph. Hence, the minimum spanning tree (MST) cost is upper bounded by $(2k-1)T^\star$. This leads to the natural idea of starting with the MST of the graph. At a high-level, we can root the tree at an arbitrary vertex, traverse it bottom-up and chop it off as soon as the total cost of the traversed sub-tree is between $[2T^\star, 4T^\star]$. This needs to be done carefully and we would refer the reader to [10] for details. Now, this can create at most k partitions of the MST, each of cost at most $4T^\star$. The rooted version uses a similar idea, but requires more care. Let us try to apply this idea to the capacitated case. A potential problem is that, we have no control over the structure of the MST. In particular, some part of the MST might be dense - it might contain a connected subtree that has small length but covers a large number of vertices, while some other parts might be sparse. Hence, cutting off the MST on the basis of length as above might end up producing infeasible trees, although the cost might be bounded. One possible idea to fix this could be to further cut off the dense subtrees and try and re-combine them with the sparse subtrees. However, it is not clear how to avoid either combining more than a constant number of subtrees or subtrees that are more than $\mathcal{O}(T^\star)$ distance away from each other and hence cannot lead to a constant factor approximation.

Our Approach: Using λ-Steiner Trees. In order to develop the intuition for our core ideas, we focus on the special case where each tree in the optimal solution *covers* exactly λ vertices (note that each tree may span more than λ vertices). We take a different approach to the problem by utilizing the concept of λ-Steiner Trees. Given a graph G and a subset of vertices R called *terminals*, a λ-Steiner tree on R is a minimum length subtree of G that contains exactly λ vertices from R. We begin with the observation that each tree in the optimal solution covers λ vertices and hence OPT can be thought of as an union of k λ-Steiner trees in G, although not necessarily of the minimum possible cost. A natural algorithm is to pick an arbitrary root vertex and construct a λ-Steiner tree. Computing the λ-MST and hence λ-Steiner trees is NP-Hard and hence we resort to the 4-approximation algorithm that essentially follows from Garg's algorithm [11]. If we are lucky, we might end up capturing an optimal tree and continue. However, in the unlucky case, the λ-Steiner tree might cover vertices from several of the optimal trees. Now if we disregard these vertices in further iterations and try to build another λ-Steiner Tree on the uncovered vertices, we might be stuck since such vertices in the union of the optimal trees might be far away from each other. Hence we cannot guarantee that a low cost λ-Steiner tree exists on these vertices.

We fix this problem by being less aggressive in the first step. We try to build as many $\lceil \lambda/2 \rceil$-Steiner trees as possible that have cost at the most $\mathcal{O}(T^\star)$ using Garg's algorithm - call such trees *good*. At some point, we might be left with vertices such that there does not exist any $\lceil \lambda/2 \rceil$-Steiner trees of low cost that can cover them - let us call them *bad vertices*. In order to cover the bad vertices, we deploy an iterative clustering procedure. We begin by applying the algorithm of Khani and Salavatipour [16] for MMTC, henceforth termed as the KS-algorithm, on the bad vertices. Note that this will return at most k trees each of cost $\mathcal{O}(T^\star)$, although we still cannot prove any lower bound on the number of vertices covered by each tree. Next we set up a bipartite matching instance with these newly formed trees on the left hand side and the good trees on the right hand side. We introduce an edge

between two trees if and only if they are separated by a distance of at the most $T^\star$. The crucial claim now is - if there exists a Hall Set in this matching instance, say S, then the number of trees in S is strictly greater than the number of trees that the optimal solution forms with the vertices covered by S. Hence, applying the KS-algorithm on the vertices in S reduces the number of trees in S without increasing the cost of each tree. This idea forms the heart of our algorithm. We apply the KS-algorithm iteratively until there is no Hall Set, at which point we can compute a perfect matching of the bad trees. Each bad tree can now be combined with a good tree to produce a tree that has cost $\mathcal{O}(T^\star)$ and contains at least $\lambda/2$ vertices each. In a nutshell, the above procedure circumvents the problem of creating too many sparse trees. It ensures that every tree is sufficiently dense - covers at least $\lceil \lambda/2 \rceil$ vertices and are of low cost. Consider a modified graph $\tilde{G}$ created by contracting the edges of the dense trees. Since the original graph is connected and each edge has length at the most $T^\star$, $\tilde{G}$ is also connected and any edge in $\tilde{G}$ has length at the most $T^\star$. We utilize these properties to distribute the vertices and ensure that the final set of trees have exactly λ vertices each and cost at the most $\mathcal{O}(T^\star)$. We note that under the assumption that every optimal solution tree has exactly λ vertices as well, the above algorithm will produce exactly k trees.

Our algorithm for the capacitated problem builds upon the above ideas. However, one major bottleneck is that we can no longer assume that each tree in the optimal solution contains exactly λ vertices. In fact, there could be trees which have a small number of vertices, say less than $\lambda/2$ - call them *light* trees and the rest *heavy*. Handling this situation requires more subtle ideas. We again start by creating $\lceil \lambda/4 \rceil$-Steiner trees of low cost as long as possible which we call good trees and as before, we shall be left with some bad trees that contain less than $\lambda/4$ vertices each, but have bounded cost. Unfortunately, the iterative refinement procedure is no longer guaranteed to produce a perfect matching, as before. However, we are able to show the following. Existence of a Hall Set even after applying the said refinement is a certificate of the fact that optimal solution contains a significant number of light trees - suppose this number is k_ℓ. Then, we add $k_\ell \cdot \lceil \lambda/2 \rceil$ dummy vertices co-located with suitably chosen bad vertices. The upshot is that, this addition ensures that each bad tree now becomes a good tree, together with the dummy vertices. Further, the total number of vertices including the dummy vertices is still bounded above by $k\lambda$. Together, this gives us that, creating trees of size exactly λ cannot result in more than k trees in the solution.

2 Capacitated Min-Max Tree Cover

In this section, we shall describe our algorithm for the CapMMTC problem and prove Theorem 3. The first step in our algorithm is to guess the value of the optimal solution - call it $T^\star$. We remove all edges from G that are of length bigger than $T^\star$ since the optimal solution can never use any such edge. The resulting graph $\hat{G}$ has, say, p connected components - call them $G_1, G_2, \cdots G_p, p \leq k$. Suppose, in the optimal solution, there are k_i trees that cover all vertices in $G_i, i = 1, 2 \cdots p$. Then, $\Sigma_{i=1}^{p} k_i = k$. For each connected component i, we run our algorithm to get at most k_i trees with cost $\mathcal{O}(T^*)$. Due to the above argument, in subsequent exposition, we shall assume that we have a connected graph $G(V, E)$ with edge lengths $\ell(e) \leq T^\star, \forall e \in E$ and there are k trees in the optimal solution that cover V such that every tree has cost at most $T^\star$ and covers at most λ vertices. There could be multiple optimal solutions. For the purpose of our discussion we pick one arbitrarily and whenever we refer to an optimal solution, we mean this particular solution. We shall divide the trees in the optimal solution into two classes for the purpose of analysis. Define a tree T in the optimal solution to be *light* if $|\text{cov}(T)| \leq \lfloor \lambda/2 \rfloor$ and *heavy* otherwise. Define k_{heavy} and k_{light} to be the number of heavy and light trees respectively. We shall prove the following theorem.

▶ **Theorem 5.** *Given a connected graph $G(V, E)$ with the edge lengths $\ell : E \to \mathbb{R}_{\geq 0}$, $\ell(e) \leq T^\star, \forall e \in E$ and non-negative integers k, λ, suppose there exists k trees $T_1, T_2, \ldots, T_k$ along with an assignment of vertices to the trees, such that T_i is a subgraph of G, $|\text{cov}(T_i)| \leq \lambda$ and $\ell(T_i) \leq T^\star$ for $i \in [k]$. Then there exists a polynomial time algorithm that finds a set of trees $T'_1, T'_2, \cdots T'_{k'}$ along with an assignment of vertices to the trees such that $k' \leq k$, for each T'_j, $\ell(T'_j) = \mathcal{O}(T^\star)$ and $|\text{cov}(T'_j)| \leq \lambda$.*

Before proceeding to the proof of the Theorem above, we show how it implies a proof for Theorem 3.

Proof of Theorem 3. We can use Theorem 5 to carry out a binary search for the correct value of $T^\star$ - the optimal solution - in the range $[0, \Sigma_{e \in E} \ell(e)]$. For a particular choice of $T^\star$, we remove all edges that are of length more than $T^\star$ and apply Theorem 5 to each of the connected components. If the total number of trees formed by our algorithm over all components is at most k, then we iterate with a guess $T^\star/2$, otherwise with $2T^\star$. The correctness follows from Theorem 5, since our algorithm will create at most k' trees for connected component V', provided optimal solution makes k' trees as well and the guessed value $T^\star$ is correct. ◀

▶ **Definition 6.** *Given an integer $\lambda > 0$, a tree T is called λ-good if $|\text{cov}(T)| \geq \lambda/4$.*

▶ **Definition 7** (λ-Steiner Trees)**.** *Given a graph $G(V, E)$ and a subset of vertices R called terminals, λ-Steiner tree is a tree which is a subgraph of G that spans exactly λ terminals. The special case of $R = V$ is called λ-MST.*

We shall be using the 2-approximation algorithm for minimum cost λ-MST from [11] as a black-box, which implies a 4-approximation for the minimum cost λ-Steiner tree problem. Our algorithm has three phases $\mathcal{A}$, $\mathcal{B}$ and $\mathcal{C}$, each making progress towards the proof of Theorem 5.

2.1 Algorithm $\mathcal{A}$: Constructing λ-good trees

Recall that the input is a graph $G = (V, E)$, with edge lengths $l_e \leq T^\star, e \in E$. Further, we assume that there exists a partition of V into k trees such that the cost of each tree is at most $T^\star$ and each tree covers at most λ vertices (with respect to some assignment). We maintain a set of covered vertices V_c throughout our algorithm. Initially $V_c = \phi$. The following is done iteratively. Pick an arbitrary vertex $v \in V \setminus V_c$ which has not been considered in any previous iteration. Construct a 4-approximate minimum cost $\lceil \lambda/4 \rceil$-Steiner Tree in G, rooted at v with the terminal set being $V \setminus V_c$. If the cost of this tree is at most $4T^\star$, then add this tree to the set λ_{good}, add all terminals spanned by this tree to V_c. At the termination of this loop, set $V_{bad} = V \setminus V_c$.

▶ **Lemma 8.** *At termination of Algorithm $\mathcal{A}$,*
1. *$V = \text{cov}(\lambda_{good}) \cup V_{bad}$, where $\text{cov}(\lambda_{good}) = \bigcup_{T \in \lambda_{good}} \text{cov}(T)$*
2. *For any tree $T \in \lambda_{good}$, $|\text{cov}(T)| = \lceil \lambda/4 \rceil$ and $\ell(T) \leq 4T^\star$*

▶ **Lemma 9.** *For any heavy tree in* OPT*, say T, at least half of the vertices covered by T in* OPT*, ie.* $\text{cov}(T)$ *are covered by the union of trees in λ_{good}*

Proof. Assume otherwise. This implies that at least $\lambda/4$ vertices covered by T are added to V_{bad} - let us call this set V'. Now, $\ell(T) \leq T^\star$ and there must exist a connected sub-tree of T that covers exactly $\lceil \lambda/4 \rceil$ vertices from V'. Hence, some iteration of Algorithm $\mathcal{A}$ must have returned a tree of cost at most $4T^\star$ covering $\lceil \lambda/4 \rceil$ vertices from V' and this tree was added to λ_{good}. Consequently, V' cannot be a subset of V_{bad} leading to a contradiction. ◀

Algorithm 1 Algorithm $\mathcal{A}$.

Input: Graph $G = (V, E), \lambda$
Output: Set of trees λ_{good} and set of vertices V_{bad}
$V_c \leftarrow \phi$, $\lambda_{good} \leftarrow \phi$, $V_{temp} \leftarrow V$
While $V_{temp} \neq \phi$
Pick $v \in V_{temp}$ arbitrarily
$T \leftarrow \lceil * \rceil \lambda/4$-Steiner tree rooted at v with terminal set defined as $V \setminus V_c$
$\text{cov}(T) \leftarrow$ terminals in $V \setminus V_c$ covered by T
If $\ell(T) \leq 4T^\star$ **Then**
Add T to λ_{good}
Update $V_{temp} \leftarrow V_{temp} \setminus \text{cov}(T), V_c \leftarrow V_c \cup \text{cov}(T)$
Else
Update $V_{temp} \leftarrow V_{temp} \setminus \{v\}$
$V_{bad} \leftarrow V \setminus V_c$
Return λ_{good} and V_{bad}

2.2 Algorithm $\mathcal{B}$: Covering V-bad vertices with λ-good trees

Recall that in the previous section, we gave algorithm to construct a set of trees λ_{good} such that for any tree $T \in \lambda_{good}$, $\ell(T) \leq 4T^\star$ and $|\text{cov}(T)| = \lceil \lambda/4 \rceil$. Further, we extracted a subset of vertices V_{bad} such that no $\lceil \lambda/4 \rceil$-Steiner tree of cost at the most $4T^\star$ exists covering these vertices. In this section, we shall give a procedure to cover the vertices in V_{bad}, either by constructing new trees that have large size but low cost, or by merging them in to the existing λ-good trees. In the following procedure, we shall be making use of the 3-approximation algorithm by Khani and Salavatipour [16] for the MMTC problem. Recall the main theorem from above paper, re-phrased to suit our notations.

▶ **Theorem 10** ([16]). *Given an undirected graph $G(V, E)$, an integer k and edge lengths ℓ, if there exists a set of k trees (subgraphs of G) $T_1, T_2, \cdots T_k$, along with an assignment of vertices to the trees, such that $\max_{i \in [k]} \ell(T_i) \leq T^\star$ and $\cup_{i=1}^{k} \text{cov}(T_i) = V$, then there exists a polynomial time algorithm that outputs trees (subgraphs of G) $T'_1, T'_2, \cdots T'_{k'}$, along with an assignment of vertices to the trees such that $\max_{j \in [k']} \ell(T'_j) \leq 3T^\star$, $\cup_{j=1}^{k'} \text{cov}(T'_j) = V$ and $k' \leq k$.*

We refer to the algorithm referred to in the above theorem as the KS-Algorithm in subsequent sections. As mentioned earlier, although the trees T'_j might share vertices and edges, the set of vertices that such trees *cover* forms a partition of V. Note that KS algorithm takes an edge weighted graph and number of trees as input.

A Bipartite Matching Instance. Our algorithm first constructs a maximum matching instance on a suitably defined bipartite graph $H = (B, Q, F)$. In the course of algorithm, Q remains fixed. However, the other partitite set B and consequently the edge set F is refined iteratively, each time giving rise to a new maximum matching instance. Our invariants will ensure that, either we end up with a perfect matching of B or conclude that there are significantly many light trees in the optimal solution, which need to be handled separately. For clarity of notation, we parameterize set B by iteration index t - define B_t to be the partite set B at iteration t. A vertex in the set B_t represents a subtree $T_{j,t}, j = 1, 2, \cdots |B_t|$ such that $\cup_{j=1}^{|B_t|} \text{cov}(T_{j,t}) = V_{bad}$. The set Q contains a vertex corresponding to each tree in $T \in \lambda_{good}$. Next, we define the edges in F_t at any iteration t. There exists an edge

$e = (T_{j,t}, T) \in F_t, T \in Q$ if and only if $d(V(T_{j,t}), V(T)) \leq T^\star$. For a fixed t, define $\mathcal{M}_t$ to be a maximum matching instance on the graph (B_t, Q, F_t). For any subset $B' \subseteq B_t$, let $\mathcal{N}(B')$ be the neighborhood set of B' in Q.

Algorithm IterRefine. We initialize by setting $B_0 = V_{bad}$ - each vertex in B_0 is a trivial tree containing a single vertex. Now we have the required setup to describe the iterative refinement procedure. At any iteration t, we solve the maximum matching instance $\mathcal{M}_t$. If $\mathcal{M}_t$ admits a perfect matching of B_t, then we terminate. Otherwise we consider the Hall Set $S \subseteq B_t$ with maximum deficiency, that is $\text{argmax}_{S \subseteq B_t}(|S| - |\mathcal{N}(S)|)$. Let set S contains the subtrees $T_{1,t}, T_{2,t}, \cdots T_{s,t}, s = |S|$. Let $G_t = (V_t, E_t)$ be defined as: $V_t = \cup_{j=1}^{s} \text{cov}(T_{j,t}), E_t = \{(u,v) | u, v \in V_t\}$ and $\ell(u,v)$ is the length of shortest path between u and v in G. We run the KS algorithm on G_t with $k = s - 1$. Suppose it returns trees $T'_{1,t}, T'_{2,t}, \cdots T'_{s',t}$, $s' \leq s - 1$. If cost of each such tree is at most $6T^*$, we define $B_{t+1} = (B_t \setminus S) \bigcup_{j=1}^{s'} \{T'_{j,t}\}$, the edge set F_{t+1} accordingly and proceed to the next iteration, else we terminate.

Algorithm 2 Algorithm IterRefine.

Input : λ_{good}, V_{bad}
Output: A maximum matching in H_t and Unmatched Set $\overline{B_m}$
Initialize the bipartite graph $H_0 = (B_0, Q, F_0), t \leftarrow 0$
While there exists a Hall Set in B_t
Pick the Hall Set $S = \text{argmax}_{S \subseteq B_t}(|S| - |\mathcal{N}(S)|)$
Apply KS-algorithm on G_t with $k = s - 1$. Let $T'_{1,t} \cdots T'_{s',t}$ be the output of the algorithm with $s' \leq s - 1$.
If $\max_{i=1}^{s'} \ell(T'_{i,t}) > 6T^*$ **Then** STOP
Else { $B_{t+1} \leftarrow (B_t \setminus S) \bigcup_{j=1}^{s'} \{T'_{j,t}\}$ and $t \leftarrow t + 1$ }
Return Maximum Matching $\mathcal{M}_t$ and unmatched subset $\overline{B_m} \subseteq B_t$

The following lemmas are crucial to prove the correctness of our algorithm. Lemma 11 is a well known result, but we give the proof for completion. We define the projection of the optimal solution on S as follows:

$$\text{OPT}\big|_S = \{T' \in \text{OPT} : \exists T_{i,t} \in S, \text{cov}(T') \cap \text{cov}(T_{i,t}) \neq \phi\}$$

▶ **Lemma 11.** *At any iteration t, let the number of trees in $\text{OPT}\big|_S$ be m. Then there exist m trees $T_i, i \in [m]$ such that $\cup_{i=1}^{m} V(T_i) = V_t, \ell(T_i) \leq 2T^*$ and T_i is a subtree of G_t for $i \in [m]$.*

Proof. Let $T'_i, i \in [m]$ be the trees in $\text{OPT}\big|_S$. Fix a tree, say T'_i. Consider the graph with each edge of T'_i doubled. It is connected and all degrees are even, hence there exists an eulerian walk, say $v_1, v_2, \ldots, v_l$. Note that vertices in this walk may be repeated. Identify a unique vertex for each $v \in V(T'_i) \cap V_t$ in this sequence. This defines a subsequence of the walk, say $v_{i_1}, v_{i_2}, \ldots, v_{i_l}$ with every vertex in $V_t \cap V(T'_i)$ occurring exactly once. This sequence defines a path in G_t since G_t is a complete graph. The total length of path is at most $2T^*$, as length of any edge in G_t is the shortest path distance between the end points in G and total length of the walk is at most $2T^*$. Repeating this argument for all trees in $\text{OPT}\big|_S$, we get that vertices in V_t can be covered by m trees of length at most $2T^*$ each. ◀

▶ **Lemma 12.** *Suppose S is the maximum deficiency Hall Set, if any, at termination of Algorithm IterRefine. Further, let k_ℓ be the number of light trees in $\text{OPT}\big|_S$. Then $|S| - |N(S)| \leq k_\ell$.*

Proof. Let k_h be the number of heavy trees in $\text{OPT}\big|_S$. We first claim that $|S| \leq k_h + k_\ell$. Assume otherwise. But then, by Lemma 11, there exists a partition of S into exactly $k_h + k_\ell$ trees in G_t with cost at most $2T^*$, by Theorem 10, the KS-algorithm applied on the Hall Set S with $k = |S| - 1$ must have produced trees with cost at most $6T^*$ and the algorithm would not have terminated.

Now, we prove that $|N(S)| \geq k_h$. Consider any heavy tree in $\text{OPT}\big|_S$, say $\widehat{T}$. By Lemma 9, at least half of the vertices in $\widehat{T}$ are covered by λ-good trees. By construction of the bipartite graph (B_t, Q), all such λ-good trees belong to the set $N(S)$, since any two vertices in the same tree of the optimal solution can be at the most $T^\star$ distance away from each other. Thus, adding up over all heavy trees in $\text{OPT}\big|_S$, the total number of vertices covered by the λ-good trees in $N(S)$ is at least $k_h \cdot \lceil \lambda/4 \rceil$. But, each λ-good tree in $N(S)$ covers exactly $\lceil \lambda/4 \rceil$ vertices. Hence, by pigeon-hole principle, $|N(S)| \geq k_h$. Combining the above two facts, we have $|S| - |N(S)| \leq k_\ell$. ◀

▶ **Corollary 13.** *Let t' be the final iteration of the Algorithm IterRefine. Then, the total number of unmatched trees in $B_{t'}$ is at most k_{light}.*

Proof. We use the fact that the size of the maximum matching in $B_{t'}$ is exactly equal to $|B_{t'}| - (|S| - |N(S)|)$, where S is the maximum deficiency Hall Set [9]. Hence, the total number of unmatched trees in $B_{t'}$ is at exactly $|S| - |N(S)| \leq k_\ell \leq k_{light}$, by Lemma 12. ◀

We proceed to the final phase of our algorithm, equipped with the above Lemmas. Recall that, there are two types of partitions in $B_{t'}$ - either matched or unmatched by $\mathcal{M}_{t'}$. First, let us consider the matched partitions - call this set B_m. For each tree T in B_m, we take its union with the λ-good tree in Q, say $\widehat{T}$ that it is matched to in $\mathcal{M}_{t'}$. More formally, we build a tree T' by connecting $T, \widehat{T}$ using the shortest path between the closest pair of vertices in T and $\widehat{T}$. We set $\text{cov}(T') = \text{cov}(T) \cup \text{cov}(\widehat{T})$. We remove $\widehat{T}$ and add the newly constructed tree T' to the set λ-good.

Finally, we consider the set of unmatched trees $\overline{B_m}$. For each such tree, we add $\lceil \lambda/2 \rceil$ dummy vertices co-located with any arbitrary vertex in the tree. We add this tree to λ_{good}. This completes the description of the algorithm.

▶ **Lemma 14.** *At termination, Algorithm $\mathcal{B}$ returns a set of trees λ_{good} that*

1. *covers all vertices in the set $V \cup V_{dummy}$.*
2. *The cost of any tree in λ_{good} is at most $11T^\star$.*
3. *$|V \cup V_{dummy}| \leq k\lambda$*

Proof. First we show that all trees in λ_{good} are indeed λ-good, that is, they cover at least $\lambda/4$ vertices. Recall that, after algorithm $\mathcal{A}$, the set Q contains only λ-good trees, by Lemma 8. At termination of algorithm $\mathcal{B}$, each tree is either formed by joining of a tree in the set Q with some tree in B_m. Hence, the tree continues to be λ-good. The only other type of tree in λ_{good} are formed by adding $\lceil \lambda/2 \rceil$ dummy vertices to an unmatched tree in $\overline{B_m}$. Hence, they are λ-good by construction. Property 1 follows from Lemma 8 and the fact that cover of trees in $B_{t'}$ is a partition of the vertices in V_{bad}. Next we prove Property 2. There are two types of trees in λ_{good}. Let us consider any tree that is formed by the union of some matched tree in B_m, say $\mathcal{T}$ and some already existing λ-good tree in Q, say $\widehat{T}$. By Lemma 8, $\ell(\widehat{T}) \leq 4T^\star$. By Lemma 11, $\ell(\mathcal{T}) \leq 6T^\star$. A tree is formed by joining these two trees by a path of cost at the most $T^\star$. So, the resultant tree has cost at the most $(4+6+1)T^\star = 11T^\star$. The second kind of tree corresponds to the unmatched vertices in $B_{t'}$ and hence their cost is bounded above by $3T^\star$, by Lemma 11. Addition of dummy vertices does not affect the cost at all.

Property 3 can be proved as follows. Recall that $|V| \leq k_{heavy}\lambda + k_{light}\lambda/2$. Now, we add $\lambda/2$ dummy vertices to each unmatched partition in $B_{t'}$, which, by Corollary 13, can be at the most $k_{light}\lambda/2$. Hence, $|V \cup V_{dummy}| \leq k_{heavy} \cdot \lambda + 2 \cdot k_{light} \cdot \lambda/2 \leq k\lambda$ ◀

The above lemma guarantees that the total number of vertices in the graph is at the most $k\lambda$. For technical reasons that would be clear in Section 2.3, we add some more dummy vertices co-located with an arbitrarily selected $v \in V$, such that the total number of vertices is an integer multiple of λ. We note at this point that we do not make any claim about the number of trees in the set λ_{good}. In fact, the number can be larger that k, the original budget. However, in the next Section, we show how to refine the trees such that number of trees is at most k, without increasing the cost by more than a constant factor and maintaining the capacity constraints. Also note that all our procedures are polynomial time.

2.3 Algorithm $\mathcal{C}$: Converting λ-good trees to feasible trees

In the previous section, we gave algorithms that ensure the property that each tree covers at least $\lambda/4$ vertices and the cost of each tree is bounded by $\mathcal{O}(T^\star)$. In this section, we prove that it is possible to construct a solution that creates at most k admissible trees given the previous solution, where k is the number of trees that the optimal solution uses to cover the vertices in the connected component $G = (V, E)$. We prove the following theorem in this section.

▶ **Theorem 15.** *Let $T_1, T_2, \ldots, T_m$ be trees returned by the Algorithms $\mathcal{A}$ and $\mathcal{B}$ such that $|\text{cov}(T_i)| \geq \lambda/c$, where $c \geq 1$, $\ell(T_i) \leq \alpha T^*$ for $1 \leq i \leq m$. If λ divides $|\cup_{i=1}^{m} \text{cov}(T_i)|$, then there exist trees $T'_1, T'_2, \ldots, T'_l$ spanning V such that $|\text{cov}(T'_i)| = \lambda, \ell(T'_i) \leq (6c+6c\alpha-2\alpha-6)T^*$ for $1 \leq i \leq l$. Further, such trees can be found in polynomial time.*

Define G_P be to be an unweighted graph with $T_1, T_2, \ldots, T_m$ as vertices. There is an edge (T_i, T_j) if $d(T_i, T_j) = d(V(T_i), V(T_j)) \leq T^*$.

▷ Claim 16. *G_P is connected. Further, the implicit cost of each unweighted edge in G_p is at most $T^\star$.*

Proof. The claim follows from the fact that we run Algorithms $\mathcal{A}$ and $\mathcal{B}$ only on connected components of $\hat{G}$ formed after removing all edges that are of length more than $T^\star$ from G. ◁

We will need the following theorem by Karaganis [15] to prove our result:

▶ **Theorem 17** ([15])**.** *Let $G = (V, E)$ be a connected simple unweighted graph. Define G^3 to be the graph on the same set of vertices and set of edges*

$$E^3 = \{(u, v) | \textit{shortest path distance between } u \textit{ and } v \textit{ in } G \textit{ is at the most } 3\}$$

Then, G^3 has a Hamiltonian path.

By the above theorem, there exists a Hamiltonian path in G_P^3. After renaming, let $T_1, T_2, \ldots, T_m$ be the order in which the vertices of G appear on that path. Then $d(T_i, T_{i+1}) \leq 3T^* + 2(\text{max diameter of any } T_i) \leq (2\alpha + 3)T^*$ for $1 \leq i \leq m-1$. We are now ready to prove the main results of this section.

▶ **Lemma 18.** *Let $T_1, T_2, \ldots, T_m$ be trees such that $|\text{cov}(T_i)| \geq \lambda/c$ for some $c > 1$, $\ell(T_i) \leq \alpha T^*$ for $1 \leq i \leq m$ and $d(T_i, T_{i+1}) \leq D$ for $1 \leq i \leq m-1$. There exist trees $T'_1, T'_2, \ldots, T'_l$ such that $d(T'_i, T'_{i+1}) \leq D$, $|\text{cov}(T'_i)| \geq \lambda$ for $1 \leq i \leq l-1$ and $\ell((T'_i) \leq (c-1)D + c\alpha T^*$ for $1 \leq i \leq l$. Further, such trees can be computed in polynomial time.*

Proof. We make $l = \lceil m/c \rceil$ trees. Let $P(T_p, T_q)$ denote the shortest path connecting trees T_p and T_q. $T'_i = \bigcup_{j=(i-1)c+1}^{min(ic,m)} (T_j \cup P(T_j, T_{j+1}))$ for $1 \leq i \leq l$. Since $T'_1, T'_2, \ldots, T'_{l-1}$ are formed by combining c trees, each covering at least λ/c vertices, $|\text{cov}(T'_i)| \geq \lambda$ vertices. Since any V' is formed by combining at most c trees, $\ell(T') \leq (c-1)D + c\alpha T^*$. Also, by construction $d(T'_i, T'_{i+1}) \leq D$ for $1 \leq i \leq l-1$ ◀

▶ **Lemma 19.** *Let $T_1, T_2, \ldots, T_m$ be trees such that $|\text{cov}(T_i)| \geq \lambda$ for $2 \leq i \leq m$, $\ell(T_i) \leq \alpha T^*$ for $1 \leq i \leq m$ and $d(T_i, T_{i+1}) \leq D$ for $1 \leq i \leq m-1$. If λ divides $|\cup_{i=1}^{m} \text{cov}(T_i)|$, then there exist trees $T'_1, T'_2, \ldots, T'_l$ such that $|\text{cov}(T'_i)| = \lambda, \ell(T'_i) \leq D + 2\alpha T^*$ for $1 \leq i \leq l$. Further, such trees can be found in polynomial time.*

Proof. We will prove this by induction on m. If $m = 1$, then λ divides $|\text{cov}(T_1)|$. We partition $\text{cov}(T_1)$ into sets of size λ arbitrarily. Cost of a tree covering each of these partitions will be at most αT^* (T_1 is one such tree). Let $m > 1$ and $|\text{cov}(T_1)| = c\lambda + p$, where $c \geq 0, 1 \leq p \leq \lambda - 1$. We first make c partitions of size λ each out of $\text{cov}(T_1)$. Cost of spanning tree covering each of these partitions is at most αT^*. We create a new tree, say T_p, by taking p remaining points of $\text{cov}(T_1)$ and any $\lambda - p$ points of $\text{cov}(T_2)$. $|\text{cov}(T_p)| = \lambda$ and $\ell(T_p) \leq \ell(T_1) + D + \ell(T_2) \leq D + 2\alpha T^*$. The remaining points form a smaller instance and by induction can be covered by trees with desired properties. This completes the induction step and proof of the theorem. ◀

Proof of Theorem 15. From the discussion above and applying Lemma 18, we get trees covering at least λ vertices and cost at most $C = (c-1)D + c\alpha T^*$. Using Lemma 19 on these trees, we get trees covering exactly λ vertices and cost at most $D + 2C$. Plugging in $D = (2\alpha + 3)T^*$ gives the desired result. ◀

Proof of Theorem 5. Plugging in $c = 4$ and $\alpha = 11$ from Lemma 14 gives a 260-factor approximation algorithm for the capacitated min max tree cover problem. ◀

3 Extension to Capacitated Rooted Tree Cover

In this section, we show how to extend Theorem 5 to the rooted setting. Recall that in CapRMMTC, in addition to the settings of CapMMTC, we given a set of root vertices, $R = \{r_1, r_2, \cdots r_k\}$. The k output trees need to be rooted at each of these root vertices. We prove the following theorem.

▶ **Theorem 20.** *Given a graph $G = (V, E)$ with the edge lengths $\ell : E \to \mathbb{R}_{\geq 0}$, a set of roots $R \subseteq V, |R| = k$ and a parameter λ, suppose there exist subtrees of G, $T_1, T_2, \cdots, T_{k'}$, where $\cup_{i=1}^{k} \text{cov}(T_i) = V$, $\max_{i \in [k']} |\text{cov}(T_i)| \leq \lambda, V(T_i) \cap R \neq \phi$ and $\ell(T_i) \leq T^\star, \forall i = 1, 2, \cdots k$. Then there exists a polynomial time algorithm that finds subtrees of G, $T'_1, T'_2, \cdots T'_{k'}, k' \leq k$ such that $\cup_{i=1}^{k'} \text{cov}(T'_i) = V$, $\max_{i \in [k']} |\text{cov}(T'_i)| \leq \lambda$, $V(T'_i) \cap R \neq \phi$ and $\ell(T'_i) \leq \beta T^\star$, where $\beta = \mathcal{O}(1)$.*

Proof. Similar to Section 2, we focus on one of the connected components, say G' obtained after deleting all edges of length larger than $T^\star$. Let $R' = \{r_1, r_2, \cdots, r_{k'}\}$ be the subset of roots belonging to this component. There exists a partition of $V' - R'$ into k' trees, each covering at most $\lambda - 1$ vertices and cost no more than $T^\star$. Using Theorem 5, we first compute subtrees of G' , $T_1, \ldots, T_{k'}$, such that $|\text{cov}(T'_i)| \leq \lambda - 1$ and cost is $\mathcal{O}(T^\star)$. We construct a bipartite graph $H' = (\mathcal{T} \cup R', E')$, where $\mathcal{T}$ contains a vertex corresponding to each tree $T'_j, j = 1, 2, \cdots, k'$. We shall abuse notations slightly and interchangeably use T'_j to denote the vertex in $\mathcal{T}$ corresponding to the tree T'_j and the actual tree itself, noting the difference

wherever necessary. $(r_i, T'_j) \in E'$ if and only if $d(r_i, T'_j) \leq T^\star$. If there exists a matching in H' such that all vertices of $\mathcal{T}$ are matched, we add the matching root vertex to T'_j. This increases the cost of a component by at most $T^\star$. Now each component contains at least one root vertex, covers at most λ vertices and costs $\mathcal{O}(T^\star)$. If all the vertices of $\mathcal{T}$ cannot be matched, there exists a set $S \subseteq T$ such that $|N(S)| < |S|$. Let $V(S) = \cup_{T'_j \in S} V(T'_j)$. In optimal solution, every vertex of $V(S)$ is contained in a tree with one of the vertices in $N(S)$ as its root. Hence, there exists $|N(S)|$ trees such that each tree covers at most $\lambda - 1$ points of $V(S)$, costs no more than $4T^\star$ and union of all the trees cover $V(S)$. We use the result of Theorem 5 to get at most $|N(S)|$ trees covering $V(S)$, such that each tree covers at most $\lambda - 1$ vertices and costs $\mathcal{O}(T^\star)$. Note that this procedure is similar to **IterRefine** defined earlier. We now have a new set of trees. We modify our bipartite graph H' by removing vertices corresponding to trees in S and replacing them by vertices corresponding to newly formed trees. We modify E' suitably. If there exists a matching in the new graph, we are done, otherwise we repeat the above procedure till we find a perfect matching. Note that the size (in terms of number of vertices) of H' decreases by at least $|S| - |N(S)| \geq 1$ after every step and hence will halt in at most k' steps. ◀

4 Conclusion

In this paper, we give the first constant factor approximation algorithms for the hard uniform capacitated versions of unrooted and rooted Min Max Tree Cover problem on a weighted graph. Our main technical contribution lies is devising an iterative refinement procedure that distributes vertices as evenly as possible among the different trees in the cover, without incurring too much cost.

Our current approximation factor is large (> 250) and although we did not attempt to optimize, it is unlikely that our approach can reduce the factor to a very small constant. Such a result requires new ideas. Perhaps somewhat surprisingly, no LPs with constant integrality gap is known, even for the uncapacitated problem. We leave open the question of finding such an LP, in particular, configuration-style LPs, which can possibly lead to a small approximation ratio. On the other hand, the only known hardness result, to the best of our knowledge, follows from the hardness of the uncapacitated version [20]. It would be interesting to prove a better lower bound for the capacitated case, if possible.

As mentioned in Section 1, there is a PTAS known for the uncapacitated rooted MMTC problem on a tree metric. We conclude with our second open question of proving a PTAS for the hard capacitated version of the same.

References

1 Noga Alon, Yossi Azar, Gerhard J Woeginger, and Tal Yadid. Approximation schemes for scheduling on parallel machines. *Journal of Scheduling*, 1(1):55–66, 1998.

2 Hyung-Chan An, Aditya Bhaskara, Chandra Chekuri, Shalmoli Gupta, Vivek Madan, and Ola Svensson. Centrality of trees for capacitated $$k$$k-center. *Math. Program.*, 154(1-2):29–53, December 2015. doi:10.1007/s10107-014-0857-y.

3 Esther M. Arkin, Refael Hassin, and Asaf Levin. Approximations for minimum and min-max vehicle routing problems. *J. Algorithms*, 59(1):1–18, 2006. doi:10.1016/j.jalgor.2005.01.007.

4 Amariah Becker, Philip N. Klein, and Aaron Schild. A PTAS for bounded-capacity vehicle routing in planar graphs. *CoRR*, abs/1901.07032, 2019. arXiv:1901.07032.

5 Amariah Becker and Alice Paul. A PTAS for minimum makespan vehicle routing in trees. *CoRR*, abs/1807.04308, 2018. arXiv:1807.04308.

6 Jaroslaw Byrka, Bartosz Rybicki, and Sumedha Uniyal. An approximation algorithm for uniform capacitated k-median problem with 1+\epsilon capacity violation. In *Integer Programming and Combinatorial Optimization - 18th International Conference, IPCO 2016, Liège, Belgium, June 1-3, 2016, Proceedings*, pages 262–274, 2016. doi:10.1007/978-3-319-33461-5_22.

7 Lin Chen, Klaus Jansen, Wenchang Luo, and Guochuan Zhang. An efficient PTAS for parallel machine scheduling with capacity constraints. In *Combinatorial Optimization and Applications - 10th International Conference, COCOA 2016, Hong Kong, China, December 16-18, 2016, Proceedings*, pages 608–623, 2016. doi:10.1007/978-3-319-48749-6_44.

8 Lin Chen and Dániel Marx. Covering a tree with rooted subtrees - parameterized and approximation algorithms. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2018, New Orleans, LA, USA, January 7-10, 2018*, pages 2801–2820, 2018. doi:10.1137/1.9781611975031.178.

9 Reinhard Diestel. *Graph Theory, 4th Edition*, volume 173 of *Graduate texts in mathematics*. Springer, 2012.

10 Guy Even, Naveen Garg, Jochen Könemann, R. Ravi, and Amitabh Sinha. Min–max tree covers of graphs. *Operations Research Letters*, 32(4):309–315, 2004. doi:10.1016/j.orl.2003.11.010.

11 Naveen Garg. Saving an epsilon: A 2-approximation for the k-mst problem in graphs. In *Proceedings of the Thirty-seventh Annual ACM Symposium on Theory of Computing*, STOC '05, pages 396–402, New York, NY, USA, 2005. ACM. doi:10.1145/1060590.1060650.

12 Nili Guttmann-Beck and Refael Hassin. Approximation algorithms for min–max tree partition. *Journal of Algorithms*, 24(2):266–286, 1997. doi:10.1006/jagm.1996.0848.

13 Mordecai Haimovich and Alexander HG Rinnooy Kan. Bounds and heuristics for capacitated routing problems. *Mathematics of operations Research*, 10(4):527–542, 1985.

14 Mong-Jen Kao. Iterative partial rounding for vertex cover with hard capacities. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 2638–2653. SIAM, 2017.

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

16 M. Reza Khani and Mohammad R. Salavatipour. Improved approximation algorithms for the min-max tree cover and bounded tree cover problems. *Algorithmica*, 69(2):443–460, June 2014. doi:10.1007/s00453-012-9740-5.

17 Hiroshi Nagamochi and Kohei Okada. Approximating the minmax rooted-tree cover in a tree. *Information Processing Letters*, 104(5):173–178, 2007. doi:10.1016/j.ipl.2007.06.012.

18 Barna Saha and Aravind Srinivasan. A new approximation technique for resource-allocation problems. *Random Struct. Algorithms*, 52(4):680–715, 2018. doi:10.1002/rsa.20756.

19 Sam Chiu-wai Wong. Tight algorithms for vertex cover with hard capacities on multigraphs and hypergraphs. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 2626–2637. Society for Industrial and Applied Mathematics, 2017.

20 Zhou Xu and Qi Wen. Approximation hardness of min-max tree covers. *Oper. Res. Lett.*, 38(3):169–173, May 2010. doi:10.1016/j.orl.2010.02.004.