Min-Max Tree Covers of Graphs*

G. Even† N. Garg‡ J. Könemann§ R. Ravi¶ A. Sinha‖

November 19, 2003

Abstract

We provide constant factor approximation algorithms for covering the nodes of a graph using trees (rooted or unrooted), under the objective function of minimizing the weight of the maximum weight tree, subject to an upper bound on the number of trees used. These problems are related to location routing and traveling salesperson problems.

Keywords: Approximation Algorithms; Graphs; Location Routing; Clustering.

*A preliminary version of this paper appears in the proceedings of the Seventh International Conference on Approximation Algorithms for Combinatorial Optimization (APPROX), 2003, with the title "Covering Graphs Using Trees and Stars". The authors would also like to thank Asaf Levin for sending us a copy of [1].

†Tel-Aviv University, Tel-Aviv, Israel. guy@eng.tau.ac.il

‡Indian Institute of Technology, Delhi, India. naveen@cse.iitd.ernet.in

§GSIA, Carnegie Mellon University, Pittsburgh, USA. jkonemann@acm.org Supported by the National Science Foundation under grant No. 0105548 and the ALADDIN Center under NSF grant No. CCR-0122581.

¶GSIA, Carnegie Mellon University, Pittsburgh, USA. ravi@cmu.edu Supported by the National Science Foundation under grant No. 0105548 and the ALADDIN Center under NSF grant No. CCR-0122581.

‖GSIA, Carnegie Mellon University, Pittsburgh, USA. asinha@andrew.cmu.edu Supported by the National Science Foundation under grant No. 0105548, the ALADDIN Center under NSF grant No. CCR-0122581 and a Carnegie Bosch Institute Fellowship. *Corresponding author.*

1 Introduction

This paper was motivated by the following "Nurse station location" problem. A hospital wanted to locate k nurses in its coverage area. Each nurse would be assigned a certain set of patients, who she would visit in her morning rounds. The objective is to figure out where to locate the nurse stations and how to assign patients to nurses so that the latest completion time is minimized.

This problem is equivalent to covering a the nodes of a metric graph with no more than k tours, so that the maximum length of a tour is minimized. Since minimum spanning trees are constant factor approximations to traveling salesperson tours, we look at covering the nodes of a graph with k trees, so that the maximum weight of a tree is minimized. If the hospital has already built its nursing stations and only wants to assign patients to nurses, we get the rooted version of this problem. The problems are defined formally in the next section.

Previous and related work. The problems studied in this paper are closely related to those studied by Arkin, Hassin, and Levin [1]. The problems they deal with include covering the nodes of a graph or a subset of the edges of a graph by paths, walks, or stars. Most of their approximation algorithms deal with minimizing the number of covering objects (e.g. paths) subject to a constraint on the cost of each covering object. They also consider unrooted versions of k-path covers and k-walk covers. The algorithms in [1] do not seem to extend to rooted versions.

Guttmann-Beck and Hassin [6] considered the unrooted version with the additional constraint that the trees have equal weights. They presented an approximation algorithm for this version with an approximation ratio of $2k - 1$.

These problems fall in the general class of "location routing" problems (see [8] for a recent survey). In the k-traveling salesperson problem, a feasible solution consists of k tours that cover the nodes, where the tours share the same depot (i.e. starting and ending point). The objective is to minimize the total length of tours. The k-traveling salesperson problem was first approximated to a constant by Frederickson, Hecht and Kim [4] (see also [7]). Recently, Fakcharoenphol, Harrelson and Rao [3] provided a constant-factor approximation algorithm for the k-traveling repairman problem, where the objective is to minimize the average waiting time of the customers.

Our results and techniques. For both the rooted and un-rooted versions of k-tree cover, we get polynomial time approximation algorithms with performance ratio 4. Both algorithms can be made strongly polynomial with a slight loss in the approximation guarantee, which becomes $4 + \epsilon$.

The algorithms are combinatorial, and rely on a matching construct to prove the approximation guarantee. These approximation algorithms algorithm imply approximation algorithms for the nurse location routing problem with an approximation ratio of $8 + \epsilon$.

Organization. We define the two versions of the problem in the next section. In Section 3, we prove that both problems are NP-hard. We provide constant factor approximation algorithms for the rooted and un-rooted versions of k-tree cover in Section 4. We conclude with some open questions in Section 5.

2 Problem definition

k-tree cover. Let $G = (V, E)$ denote an undirected graph with positive integral edge weights $w : E \to \mathbb{N}^+$. A *tree cover* of a graph $G = (V, E)$ is a set $\mathcal{T}$ of trees $\{T_i\}_i$ such that $V = \bigcup_{i=1}^{k} V(T_i)$. The weight of a tree T is defined by $w(T) = \sum_{e \in T} w(e)$. The *cost* of a tree cover $\mathcal{T}$ is $\max_{T_i \in \mathcal{T}} w(T_i)$.

Note that trees in a tree cover may share nodes and even edges. The goal in the *min-max k-tree cover* problem is to find a minimum cost tree cover consisting of at most k trees.

R-Rooted tree cover. Let $R \subset V$ denote a set of *roots*. An *R-rooted tree cover* of a graph $G = (V, E)$ is a tree cover $\mathcal{T}$, where each tree $T_i \in \mathcal{T}$ has a distinct root in R.

As before, trees in an R-rooted tree cover may share nodes and edges. In particular, the root of T_i may be in T_j, but the roots of T_j and T_i must be distinct. Given an edge weighted graph G and a set R of roots, the *min-max R-rooted tree cover* problem is to find a minimum cost R-rooted tree cover of G.

3 Hardness

In this section, we show that both problems are NP-complete. We begin by showing the NP-completeness of R-rooted tree cover, by reducing BIN-PACK to it. An instance of BIN-PACK consists of (i) a set U of elements, where the size of an element $u \in U$ is s_u, (ii) k bins, and (iii) a positive bin capacity B. The problem is to determine if there is a partition of U into k parts $U_1, \ldots, U_k$ such that for every $i = 1, \ldots, k$, we have $\sum_{u \in U_i} s_u \leq B$. This was shown to be NP-hard in [5].

Theorem 1 *The min-max R-rooted tree cover problem is NP-complete.*

Proof Given an instance $\Pi = \langle U, \{s_u\}_u, k, B\rangle$ of BIN-PACK, we transform it to an instance of R-rooted tree cover as follows. We create a complete bi-partite graph $G(\Pi)$ with a vertex set $R \cup U$, where R is a set of k new nodes $R = \{r_1, \ldots, r_k\}$. For every r_i and every $u \in U$, the weight of an edge $e = (r_i, u)$ is set to $w(e) = s_u$. We complete $G(\Pi)$ into a metric space (i.e. complete graph) $K(\Pi)$ by taking the metric completion of the edge weighted graph $G(\Pi)$. We designate R to be the set of roots, and ask if there is an R-rooted tree cover of $K(\Pi)$ of cost no more than B.

It is immediate that every bin packing induces an R-rooted tree cover of the same cost. Conversely, every R-rooted tree cover can be transformed in polynomial time into a solution of the same cost for BIN-PACK, as follows. For every tree T_i rooted at r_i, replace all edges of the form (u_j, r_k) (where $r_k \neq r_i$) by an edge (u_j, r_i). Each such exchange is between edges of equal cost, and therefore the solution cost does not change. At the end of this procedure, each element in U is assigned to a unique root in R, which specifies a BIN-PACK solution of the same cost. Since an R-rooted tree cover of cost B gives a solution of cost B for BIN-PACK, the theorem follows. □

Theorem 1 can be easily extended to deal with k-tree covers. One can simply replace a tree T in $K(\pi)$ with a star S rooted at any node $r \in R$ that covers the same set of nodes in U. Moreover, $w(T) \geq w(S)$. Hence the restriction that trees are rooted at nodes of R can be met without increasing the cost of the tree cover. We therefore also have the following theorem.

Theorem 2 *The min-max k-tree cover problem is NP-complete.*

4 Clustering into trees

4.1 R-rooted tree cover

4.1.1 Algorithm.

In this section we present a 4-approximation algorithm for the min-max R-rooted tree cover problem. A strongly polynomial version of this algorithm has an approximation ratio of $(4 + \varepsilon)$.

The approximation algorithm is based on Algorithm Rooted-Tree-Cover, which is given (i) a graph $G = (V, E)$ with edge weights $w(e)$, (ii) a set $R = \{r_1, \ldots, r_k\}$ of k roots, and (iii) a bound B on the weight of each tree. Algorithm Rooted-Tree-Cover either returns a proof that B is too small (i.e., $B < B^*$, the minimum cost of a tree cover) or finds an R-rooted tree cover of cost at most $4B$. By applying binary search, a 4-approximation algorithm is obtained. In Section 4.1.3 we discuss how to derive a strongly polynomial algorithm.

Algorithm 1 Rooted-Tree-Cover(G, R, B) - Compute an R-rooted tree cover of G with cost at most $4B$.

1: Remove all edges of weight greater than B.
2: $M \leftarrow$ MST of graph obtained from G by contracting roots in R to a single node.
3: $\{T_i\}_i \leftarrow$ forest obtained from M by un-contracting roots in R.
4: Edge-decompose each tree T_i into trees $\{S_j^i\}_j + L_i$ such that $w(S_i^j) \in [B, 2B)$, for every j, and $w(L_i) < B$.
5: Try to match the trees $\{S_j^i\}_{i,j}$ to roots, subject to the constraint that a tree S_i^j may be matched only to roots of distance at most B from it.
6: If not all trees are matched, then **return** fail: "B is too low".
7: If every tree is matched, then **return** success: set of trees where each tree consists of S_j^i, its matched root r, and the leftover tree L (if any) that contains the root r.

A listing outlining Algorithm Rooted-Tree-Cover appears in Algorithm 1. We now explain each step in detail (see Fig. 4.1.1 for an example).

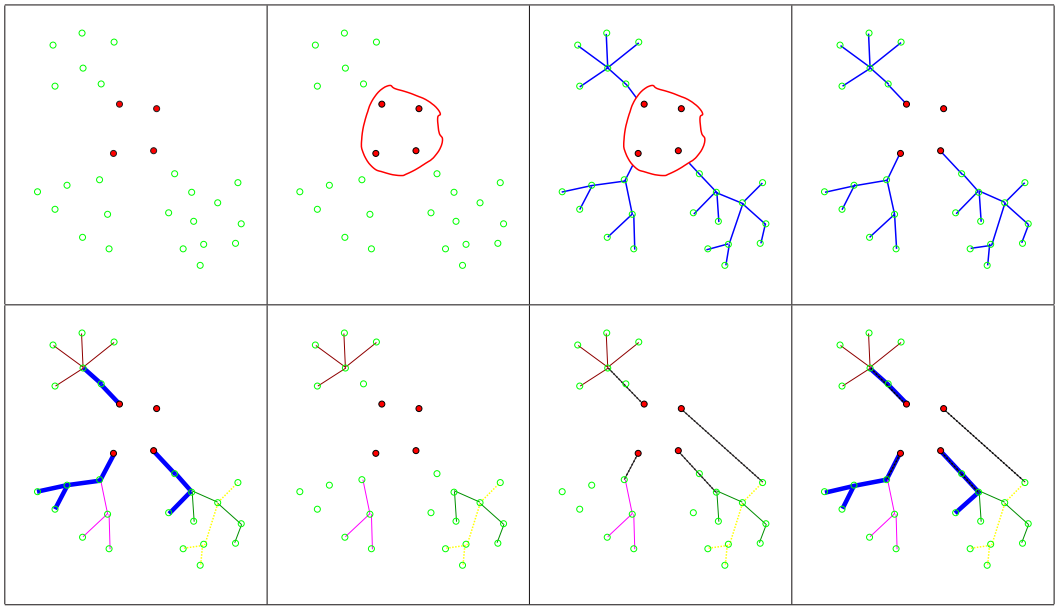


Figure 1: Example of an execution of Algorithm 1 (read from left to right): (1) The input; roots are denoted by filled nodes. (2) Contraction of the roots. (3) MST of the contracted graph. (4) Un-contracting the graph. (5) Edge decomposition of each tree; leftover subtrees $\{L_i\}_i$ are denoted by dark thick edges. (6) The non-leftover subtrees $\{S_j^i\}_{i,j}$. (7) matching of the non-leftover subtrees to roots. (8) The final trees consisting of a leftover tree, a matcher tree, and a matching edge.

The algorithm begins by removing all edges of weight greater than B, since they obviously cannot be used. If as a result of deleting heavy edges (i.e., $w(e) > B$) there exists a node that is no longer connected to R, then obviously $B \leq B^*$. To keep the description simple, we assume that the graph remains connected even after the heavy edges are deleted. In Line 2, the roots in R are contracted to a single node. Namely, (i) a new node u_R is introduced and the roots in R are removed, (ii) for

every root $r_i \in R$, an edge (v, r_i) induces an edge (v, u_R), and (iii) $w(v, u_R) = \min_i w(v, r_i)$. The algorithm computes a minimum spanning tree (MST) in the contracted graph. In Line 3, the MST is broken into a set $\{T_i\}_i$ of k disjoint trees by un-contracting the nodes in R. This un-contraction means that an edge (v, u_R) is mapped to an edge (v, r_i), where $w(v, U_R) = w(v, r_i)$. Note that, by construction, every tree T_i is rooted at r_i. In Line 4, the edge set of every tree T_i is decomposed into subtrees $\{S_j^i\}_j + L_i$. The subtrees may share nodes but not edges. The weight of every sub-tree S_j^i is in the range $[B, 2B)$ and there is perhaps a leftover tree L_i whose weight is less than B. We elaborate below how this edge decomposition is performed. Note that if L_i exists, then it contains the root r_i. However, the root may belong to other subtrees as well (since subtrees are edge disjoint but not necessarily vertex disjoint). In Line 5, a bi-partite graph is constructed as follows. One side of the vertex set is R and the other side consists of nodes representing the trees $\{S_j^i\}_{i,j}$. An edge connects a root r and a tree S_j^i if the distance between S_j^i and r is at most B. A maximum matching is then computed in this bi-partite graph. The algorithm considers now two cases: If the maximum matching does not match all the sub-trees, then in Line 6, the algorithm reports a failure by returning the statement that B is too small. If the maximum matching matches all the sub-trees, then the algorithm returns the set of trees where each tree consists of a subtree S_j^i, the root r matched to the subtree S_j^i, a shortest path from the root r to S_j^i, and the leftover tree L (if any) that contains the root r.

We now elaborate on how the edge set of every tree is decomposed in Line 4. Consider a tree T_i rooted at r. For a node $v \in V(T_i)$ let T_v denote the rooted subtree hanging from v. Consider an edge $e = (u, v)$ where u is the parent of v. The subtree T_e is the subtree that contains three parts u, T_v, and the edge (u, v). The weight $w(T_e)$ of a subtree T_e is the sum of the edge weights in T_e. Given the threshold value B, depending on $w(T_e)$, a subtree T_e is defined as *light, medium* or *heavy* as follows. If $w(T_e) \geq 2B$, then T_e is heavy. If $w(T_e) < B$, then T_e is light. If $w(T_e) \in [B, 2B)$, then T_e is medium. The decomposition algorithm proceeds by splitting away subtrees. Recall that subtrees may share nodes, hence the definition of splitting T' away from T_i means: (i) designate T' as a new part, (ii) remove the edges of T' from T_i, and (iii) let T_i now contain only nodes and edges that are still connected to the root of T_i. Note that, when $T_{(u,v)}$ is split away from T_i, the node u remains a node in T_i so $T_{(u,v)}$ and the remaining tree share the node u.

One can always split away medium subtrees from the (remaining) tree. Since such medium weight subtrees are split away whenever possible, we now focus on the case that subtrees are either light or heavy. We refer to a node v as heavy (resp., light) if T_v is heavy (resp., light). If every

subtree is either heavy or light, let v denote a a heavy node, all the children of which are light. We bunch edges $e_1, e_2, \ldots$ emanating from v to children of v until the first time the cumulative weight of the trees hanging from these edges exceeds B. We then split away the subtree $\bigcup_i T_{e_i}$ (note that this tree is a medium subtree since $w(T_{e_i}) < B$, for every i). The decomposition stops as soon as the weight of the remaining tree is less than B. If upon termination the edge set of T_i is not empty, then T_i is declared as a leftover tree L_i. Note that in this case the root of the leftover tree L_i is r (where $r \in R$ is the root of the tree T_i) .

4.1.2 Correctness and approximation ratio.

In this section we prove two lemmas: Lemma 1 proves the correctness of the algorithm and Lemma 2 proves its approximation ratio. Let B^* be the minimum cost of a tree cover of G.

Lemma 1 *If Algorithm Rooted-Tree-Cover returns "B is too low", then $B^* > B$.*

Proof We prove the contrapositive, namely, if $B \geq B^*$ then there exists a matching in the bi-partite graph that matches every subtree in $\{S_j^i\}_{i,j}$ to a root in R. The existence of such a matching is equivalent by Hall's Marriage Theorem [2] to the condition that, for every subset $\mathcal{S}$ of trees from $\{S_j^i\}_{i,j}$, the neighbor set $N(\mathcal{S})$ of $\mathcal{S}$ satisfies $|N(\mathcal{S})| \geq |\mathcal{S}|$.

Consider a subset $\mathcal{S}$ of trees from $\{S_j^i\}_{i,j}$. Every tree $S \in \mathcal{S}$ satisfies $w(S) \in [B, 2B)$. Hence, $w(\mathcal{S}) \geq B \cdot |\mathcal{S}|$.

Consider an optimal R-rooted tree cover $\mathcal{T}^* = \{T_1^*, \ldots, T_k^*\}$. Let $\mathcal{T}^*(\mathcal{S})$ denote the subset of trees of $\mathcal{T}^*$ that have a non empty intersection with a tree from $\mathcal{S}$. Namely, $T_i^* \in \mathcal{T}^*(\mathcal{S})$ iff there exists a tree $S \in \mathcal{S}$ such that $S \cap T_i^*$ is non-empty. Since $B \geq B^*$, there is an edge in the bi-partite graph between a tree S_j^i and r if the tree T_ℓ^* rooted at r intersects the tree S_j^i. Hence $|N(\mathcal{S})| \geq |\mathcal{T}^*(\mathcal{S})|$ and it suffices to prove that $|\mathcal{T}^*(\mathcal{S})| \geq |\mathcal{S}|$. Note that the weight of $\mathcal{T}^*(\mathcal{S})$ satisfies $w(\mathcal{T}^*(\mathcal{S})) \leq B^* \cdot |\mathcal{T}^*(\mathcal{S})|$.

Every node in $\bigcup \mathcal{S}$ is connected by edges in $\bigcup \mathcal{T}^*(\mathcal{S})$ to a root. Recall that every edge in S_j^i is also an edge in the MST M (from Line 2). Let M' denote the subgraph obtained from the MST M by deleting edges in $\bigcup \mathcal{S}$ and then adding edges in $\bigcup \mathcal{T}^*(\mathcal{S})$. Every vertex is connected in M' to a root, hence, the subgraph M' is connected if the roots are contracted. It follows that $w(M') \geq w(M)$, and hence, $w(\mathcal{T}^*(\mathcal{S})) \geq w(\mathcal{S})$. We conclude that $B^* \cdot |\mathcal{T}^*(\mathcal{S})| \geq w(\mathcal{T}^*(\mathcal{S})) \geq w(\mathcal{S}) \geq B \cdot |\mathcal{S}|$. Since $B^* \leq B$, it follows that $|\mathcal{T}^*(\mathcal{S})| \geq |\mathcal{S}|$. Hence, Hall's condition holds, and the lemma follows. □

The following lemma proves that the approximation ratio is 4.

Lemma 2 *When successful, Algorithm Rooted-Tree-Cover finds an R-rooted tree cover of cost at most $4B$.*

Proof By construction, each tree returned by the algorithm has a distinct root from R and every node belongs to at least one tree. The weight of each tree equals the weight of the tree S_j^i (which is bounded by $2B$), the weight of the path from the root r to a node in S_j^i (which is bounded by B), and the weight of the leftover tree (which is bounded by B). It follows that the weight of every tree is less than $4B$, and the lemma follows. □

Note that a path from a root r to a subtree S_j^i may contain edges and nodes that also belong to other trees. Hence, when successful, Algorithm Rooted-Tree-Cover covers the graph with trees, but these trees are not disjoint.

4.1.3 Strongly polynomial algorithm.

Let $n = |V|$ and consider an $\varepsilon > 0$. Our goal is to find a $(4+\varepsilon)$-approximation algorithm that is polynomial in n and in $\log \frac{1}{\varepsilon}$. Sort the edge weights, let $w_1 < w_2 < \cdots < w_m$ denote the sorted edge weights. Obviously $B^* < n \cdot w_m$. If Algorithm Rooted-Tree-Cover reports that $B < B^*$ for $B = w_m$, then the weight of all edges of weight at most $w_m/(n^2/\varepsilon)$ is less than $\varepsilon \cdot B^*$. Hence, we may contract all these edges, and consider only the remaining edges of weight at least $\varepsilon \cdot w_m/n^2$. Now binary search within the range $[\varepsilon \cdot w_m/n^2, n \cdot w_m]$ is strongly polynomial.

If Algorithm Rooted-Tree-Cover does not fail with $B = w_m$ we do the following. Let i denote an index such that (i) Algorithm Rooted-Tree-Cover reports $B < B^*$ for $B = w_i$, and (ii) Algorithm Rooted-Tree-Cover finds an R-rooted tree cover of cost at most $4 \cdot w_{i+1}$. Hence $B^* \in (w_i, 4 \cdot w_{i+1}]$. If $w_{i+1}/w_i \leq \frac{n^2}{\varepsilon}$, binary search within the range $[w_i, w_{i+1}]$ is strongly polynomial. Otherwise, let $w' = \frac{n^2}{\varepsilon} \cdot w_i$. Run Algorithm Rooted-Tree-Cover with $B = w'$. If the algorithm finds an R-rooted tree cover of cost $4w'$, binary search within the range $[w_i, w']$ is strongly polynomial. If the algorithm reports that $w' < B^*$, the weight of all edges of weight at most w_i is bounded by $n^2 \cdot w_i \leq \varepsilon \cdot B^*$. Hence, we may contract all these edges, and consider only the remaining edges of weight at most $4 \cdot w_{i+1}$. Now binary search within the range $[w_{i+1}, 4 \cdot w_{i+1}]$ is strongly polynomial.

By combining Lemmas 1, 2, and the above discussion we conclude with the following theorem. Note that if edge weights are bounded by a polynomial in n, then a 4-approximation algorithm follows.

Theorem 3 *For every ε, there is a $(4+\varepsilon)$-approximation algorithm for min-max rooted tree cover*

Algorithm 2 Tree-Cover(G, k, B) - Compute an k-tree cover of G with cost at most $4B$.

1: Remove all edges of weight greater than B. Let $\{G_i\}_i$ denote the connected components after deleting heavy edges.
2: $MST_i \leftarrow$ MST of G_i.
3: $k_i \leftarrow \lfloor \frac{w(MST_i)}{2B} \rfloor$.
4: If $\sum_i (k_i + 1) > k$ then **Return** fail: "B is too low".
5: Edge-decompose each tree MST_i into at most (k_i+1) trees $\{S_j^i\}_j + L_i$ such that $w(S_i^j) \in [2B, 4B)$, for every j, and $w(L_i) < 2B$.
6: **Return** success: set of trees $\{S_j^i\}_{i,j} \cup \{L_i\}_i$.

that runs in time polynomial in the size of G *and* $\log(\frac{1}{\varepsilon})$.

4.2 k-tree cover

In this section we present a 4-approximation algorithm for the k-tree cover problem. A strongly polynomial version of this algorithm has an approximation ratio of $(4 + \varepsilon)$.

4.2.1 Algorithm.

A listing of Algorithm Tree-Cover appears as Algorithm 2. The input consists of (i) a graph $G = (V, E)$ with positive integral edge weights $w(e)$, (ii) a bound k on the number of trees allowed in the cover, and (iii) a bound B on the weight of each tree in the cover. The algorithm returns either "fail" (meaning that B is too small), or "success" with a tree cover the cost of which is bounded by $4B$.

As in Algorithm Rooted-Tree-Cover, Algorithm Tree-Cover begins by removing edges of weight bigger than B. The removal of heavy edges may render G unconnected; we denote the connected components by $\{G_i\}_i$. In Line 2, a minimum spanning tree MST_i is computed for each component G_i. In Line 3, an estimate k_i of the number of trees needed to cover G_i is computed. In Line 4, the algorithm returns with "fail" if the estimates are too small. This means that the algorithm has a proof that the cost of an optimal k-tree cover of G is greater than B. In Line 5, each tree MST_i is edge decomposed to at most (k_i+1) subtrees. Each subtree is of cost at most $4B$. The edge-decomposition procedure is the same procedure that is used in Line 4 in Algorithm Rooted-Tree-Cover (with the threshold $2B$ instead of B). In Line 6, a tree cover consisting of at most k trees is returned. The cost of the returned tree cover is at most $4B$.

Note that the edge-decomposition procedure decomposes MST_i into at most $k_i + 1$ trees. By setting a threshold of $2B$ it follows that the weight of each tree S_j^i is at least $2B$ and at most $4B$. It

follows that the number of trees $\{S_j^i\}_j$ obtained when decomposing MST_i is at most k_i. Together with the the leftover tree L_i (if it exists) we obtain at most $k_i + 1$ trees.

4.2.2 Correctness and approximation ratio.

Let B^* denote the cost of a min-max k-tree cover of G. Let $\mathcal{T}^* = \{T_1^*, \ldots, T_k^*\}$ denote an optimal k-tree cover. If $B^* \leq B$, then $\mathcal{T}^*$ uses edges of weight no greater than B. Let k_i^* denote the number of trees in $\mathcal{T}^*$ that contain nodes of G_i.

Lemma 3 *If $B^* \leq B$ then $k_i + 1 \leq k_i^*$, for every i.*

Proof For simplicity, let $T_1^*, \ldots T_{k_i^*}^*$ denote the trees that cover G_i. By adding at most $k_i^* - 1$ edges, one can connect these k_i^* trees to obtain a tree that spans G_i. Since the cost of each such edge is at most B, we obtain: $\sum_{j=1}^{k_i^*} w(T_i^*) + (k_i^* - 1) \cdot B \geq w(MST_i)$. Since $w(T_i^*) \leq B$, we obtain $k_i^* \geq \frac{w(MST_i)}{2B} + \frac{1}{2}$. The lemma follows because $k_i \leq \frac{w(MST_i)}{2B}$. □

Lemma 3 immediately implies the following lemma.

Lemma 4 *If Algorithm Tree-Cover returns "B is too low", then $B^* > B$.*

We conclude with the following theorem. Note that a 4-approximation algorithm is obtained if the edge weights are polynomial.

Theorem 4 *For every ε, there is a $(4+\varepsilon)$-approximation algorithm for min-max tree cover that runs in time polynomial in the size of the graph and in $\log(\frac{1}{\varepsilon})$.*

Proof When $B \geq B^*$, Lemma 4 implies that Algorithm 2 is successful and a k-tree cover of cost at most $4B$ is computed. A strongly-polynomial binary search along the lines of Section 4.1.3 completes the proof. □

5 Open questions

By doubling edges in each tree, we can transform each tree into a tour, the weight of which is at most twice the weight of the tree. Hence, our algorithms for k-tree cover immediately yield constant factor approximation algorithms for the "Nurse station location" problem (which one might call the k-tour cover problem) that motivated this research. However, it may be possible to obtain improved approximation factors by attacking the k-tour cover problem directly, instead of going via k-tree cover.

References

[1] E. Arkin, R. Hassin and A. Levin. Approximations for minimum and min-max vehicle routing problems. *Manuscript*, 2003.

[2] R. Diestel. *Graph Theory*, Springer-Verlag, Berlin, 2000.

[3] J. Fakcharoenphol, C. Harrelson and S. Rao. The k-traveling repairman problem. In *Proceedings of the* 10^{th} *Annual ACM-SIAM Symposium on Discrete Algorithms*, 655-664, 2003.

[4] G.N. Frederickson, M.S. Hecht and C.E. Kim. Approximation algorithms for some routing problems. *SIAM J. Computing* 7:178-193, 1978.

[5] M. Garey and D. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W.H. Freeman, San Francisco, 1979.

[6] N. Guttmann-Beck, R. Hassin. Approximations Algorithms for Min-Max Tree Partition, *Journal of Algorithms* 24: 266-286, 1997.

[7] M. Haimovich, A. Rinnooy Kan and L. Stougie. Analysis of heuristics for vehicle routing problems. In B.L. Golden and A.A. Assad (editors), *Vehicle Routing: Methods and Studies*, North-Holland, 1988.

[8] P. Toth and D. Vigo (editors). *The Vehicle Routing Problem*, SIAM monographs on discrete mathematics and applications, 2002.