# Covering Graphs using Trees and Stars*

G. Even[1], N. Garg[2], J. Könemann[3], R. Ravi[3], and A. Sinha[3]

[1] Tel-Aviv University, Tel-Aviv, Israel. guy@eng.tau.ac.il
[2] Indian Institute of Technology, Delhi, India. naveen@cse.iitd.ernet.in
[3] GSIA, Carnegie Mellon University, Pittsburgh, USA.
jkonemann@acm.org, ravi@cmu.edu, asinha@andrew.cmu.edu

**Abstract.** A tree cover of a graph $G$ is defined as a collection of trees such that their union includes all the vertices of $G$. The cost of a tree cover is the weight of the maximum weight tree in the tree cover. Given a positive integer $k$, the $k$-tree cover problem is to compute a minimum cost tree cover which has no more than $k$ trees. Star covers are defined analogously. Additionally, we may also be provided with a set of $k$ vertices which are to serve as roots of the trees or stars. In this paper, we provide constant factor approximation algorithms for finding tree and star covers of graphs, in the rooted and un-rooted versions.

## 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 last completion time is minimized.

This problem is equivalent to covering 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 graph with no more than $k$ spanning trees, so that the maximum weight of a tree is minimized. A variant is when the nurse has to return to her station before visiting each patient, in order to pick up equipment and supplies necessary for that patient. In that case, we get the problem of covering the graph with stars. If the hospital has already built its nursing stations and only wants to assign patients to nurses, we get the rooted versions of these problems. 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

* J. Könemann, R. Ravi and A. Sinha are supported by the National Science Foundation under grant No. 0105548 and the ALADDIN Center under NSF grant No. CCR-0122581. A. Sinha is also supported by a Carnegie Bosch Institute Fellowship.

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.

These problems fall in the general class of "vehicle routing" problems (see [16] 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 that 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 [8] (see also [11]). Recently, Fakcharoenphol, Harrelson and Rao [7] 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.

The problems we study can also be viewed as members of the family of "clustering" problems. We are partitioning the vertices of the graph into clusters, where the weight of a cluster is the weight of the cheapest spanning tree or star in the cluster. Other versions of clustering studied recently include the cases where the weight of a cluster is the cost of a clique on it [2, 10], the maximum cluster diameter (also called $k$-center) [6, 12] and the sum of radii of clusters [3].

**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.

We use LP rounding to provide a $(4, 4)$ bi-criteria approximation algorithm for un-rooted $k$-star cover. That is, our algorithm outputs a solution which covers the graph with no more than $4k$ stars, and the cost of the solution is no more than four times the cost of an optimal solution which uses no more than $k$ stars. Finally, we show that the rooted version of $k$-star cover reduces to a scheduling problem studied by Shmoys and Tardos [15]. This immediately yields a 2-approximation algorithm for this problem.

For the unrooted $k$-star problem, Levin [13] suggested an improvement of our $(4, 4)$ bi-criteria approximation to a $(3, 3)$ bi-criteria algorithm; this improvement is based on the minimum star cover approximation algorithm from [1].

**Organization.** We define the four versions of the problem in the next section. In Section 3, we prove that all four 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 deal with $k$-star cover in Section 5. We conclude with some open questions in Section 6.

## 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 *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.

**Rooted $k$-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 $G$.

**Star covers.** Star cover problems are defined over complete graphs (i.e. finite metric spaces). The goal is to cover the vertices of the graph with stars so that the maximum weight of a star is minimized.

A *star cover* is a cover of the vertex set of a graph by stars. The *cost* of a star cover is the weight of the heaviest star in the cover. In the *min-max $k$-star cover* problem, the goal is to find a minimum cost star cover using at most $k$ stars. Let $R = \{r_i\}_i$ denote a set of roots. In the *min-max $R$-centered star cover* problem the goal to find a minimum cost star cover $\mathcal{S} = \{S_r\}_{r \in R}$ such that, for every $r \in R$, the center of $S_r$ is $r$.

## 3 Hardness

In this section, we show that all four problems are NP-complete. We begin by showing the NP-completeness of $R$-centered star cover, and then extend the result to the other three problems.

We show the NP-completeness of $R$-centered star 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 [9].

**Theorem 1.** *The min-max $R$-centered star 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 $R$-centered star cover as follows. We create a complete bipartite 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)

**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 $\{S^i_j\}_j + L_i$ such that $w(S^j_i) \in [B, 2B)$, for every $j$, and $w(L_i) < B$.
5: Try to match the trees $\{S^i_j\}_{i,j}$ to roots, subject to the constraint that a tree $S^j_i$ 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 $S^i_j$, its matched root $r$, and the leftover tree $L$ (if any) that contains the root $r$.

$K(\Pi)$ 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 $R$-centered star cover of $K(\Pi)$ of cost no more than $B$.

It is immediate that every bin packing induces an $R$-centered star cover of the same cost. Conversely, every $R$-centered star cover induces a partition of $U$, where a bin size equals the weight of the corresponding star. Since an $R$-centered star cover of cost $B$ gives a solution of cost $B$ for BIN-PACK, the theorem follows.

The following theorem can be proved by converting in polynomial time an optimal solution to any of the three other problems to an $R$-star cover at the same cost.

**Theorem 2.** *The following problems are NP-complete: min-max $R$-tree cover, min-max $k$-tree cover, and min-max $k$-star cover.*

## 4 Clustering into trees

### 4.1 *R*-rooted tree cover

**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 $R$ of $k$ roots, and (iii) a bound $B$ on the weight of each tree. Algorithm Rooted-Tree-Cover either returns with 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 we discuss a how to derive a strongly polynomial algorithm.

A listing outlining Algorithm Rooted-Tree-Cover appears in Algorithm 1. We now explain each step in detail. 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 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, and 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 $R$. Note that, by construction, every tree $T_i$ is rooted at a root of $R$. In Line 4, the edge set of every tree $T_i$ is decomposed into subtrees $\{S^i_j\}_j + L_i$. The subtrees may share nodes but not edges. The weight of every sub-tree $S^i_j$ 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. 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 $\{S^i_j\}_{i,j}$. An edge connects a root $r$ and a tree $S^i_j$ if the distance between $S^i_j$ 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 $S^i_j$, the root $r$ matched to the subtree $S^i_j$, a shortest path from the root $r$ to $S^i_j$, 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 $v$ 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. If every subtree is either heavy or light, let $v$ denote 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 $T_i$) .

**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 Algo. 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 $\{S^i_j\}_{i,j}$ to a root in $R$. The existence of such a matching is equivalent by Hall's Marriage Theorem [5] to the condition that, for every subset $\mathcal{S}$ of trees from $\{S^i_j\}_{i,j}$, the neighbor set $N(\mathcal{S})$ of $\mathcal{S}$ satisfies $|N(\mathcal{S})| \geq |\mathcal{S}|$.

Consider a subset $\mathcal{S}$ of trees from $\{S^i_j\}_{i,j}$. 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 are stabbed by trees 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. Note that there is an edge in the bi-partite graph between a tree $S^i_j$ and $r$ if the tree $T^*_\ell$ rooted at $r$ intersects the tree $S^i_j$. 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 $S^i_j$ 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 $S^i_j$ (which is bounded by $2B$), the weight of the path from the root $r$ to a node in $S^i_j$ (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 $S^i_j$ 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.

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

```
Remove all edges of weight greater than B. Let {G_i}_i denote the connected components after deleting heavy edges.
MST_i ← MST of G_i.
k_i ← ⌊w(MST_i)/(2B)⌋.
If ∑_i (k_i + 1) > k then Return fail: "B is too low".
Edge-decompose each tree MST_i into at most (k_i + 1) trees {S^i_j}_j + L_i such that w(S^j_i) ∈ [2B, 4B), for every j, and w(L_i) < 2B.
Return success: set of trees {S^i_j}_{i,j} ∪ {L_i}_i.
```

**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 $4 \cdot w_{i+1}$. 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 polynomial, then a 4-approximation algorithm follows.

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

**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 $S^i_j$ is at least $2B$ and at most $4B$. It follows that the number of trees $\{S^i_j\}_j$ 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.

**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 $\varepsilon$, 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 completes the proof.

## 5 Clustering into stars

In this section we discuss the un-rooted and rooted $k$-star cover problem. Here, we are given an undirected complete graph $G = (V, E)$, a metric $c$ on the edges of $E$ and a parameter $k > 0$. In the rooted version, we are also given a set of root nodes $R$ with $|R| = k$. For a set $S \subseteq V$ and a vertex $r \in V$, let $c(S, r)$ be the cost of the star rooted at $r$ and spanning $S$, i.e. $c(S, r) = \sum_{v \in S} c_{rv}$.

In the un-rooted version, we are supposed to partition the vertex set into $k$ subsets $S_1, \ldots, S_k$ and place $k$ roots $r_1, \ldots, r_k$ such that $\max_{1 \leq i \leq k} c(S_i, r_i)$ is as small as possible. In the rooted version, $r_1, \ldots, r_k$ are given.

### 5.1 Un-rooted $k$-star cover

We use linear programming techniques similar to those used in facility location problems [4, 14] to solve this problem. We first give a natural integer programming formulation. We have a variable $y_i$ for each $i \in V$ that indicates the number of stars that are rooted at node $i$. For each pair of nodes $i$ and $j$, variable $x_{ij}$ has value one if node $j$ is in a star rooted at node $i$ and zero otherwise. As in the previous section, we guess the maximum star cost $B$ of an optimum solution, and try to minimize the number of stars needed to cover the graph.

$$
\begin{aligned}
\min \quad & \sum_{i \in V} y_i & & & (IP_B) \\
\text{s.t} \quad & \sum_{i \in V : c_{ij} \leq B} x_{ij} \geq 1 & & \forall j \in V & (1) \\
& x_{ij} \leq y_i & & \forall i, j \in V & (2) \\
& \sum_{j \in V} x_{ij} c_{ij} \leq B \cdot y_i & & \forall i \in V & (3) \\
& x_{ij} \in \{0, 1\}, y_i \in \mathbb{N} & & \forall i, j \in V & (4)
\end{aligned}
$$

Constraints (1) ensure that each node $j \in V$ is assigned to a root; (2) makes sure that node $i$ is marked as a root if a node $j$ is assigned to it; (3) bounds the cost of each of the star rooted at node $i$ by $B$; and finally (4) sets integrality constraints. We denote the linear programming relaxation of $(IP_B)$ by (LP). Let $(x, y)$ be a solution to (LP). The following observation is immediate.

**Lemma 5.** *Suppose there exists a solution to the un-rooted $k$-star cover problem with value $B$. Then $(LP_B)$ has a solution $(x, y)$ such that $\sum_{i \in V} y_i \leq k$.*

Our algorithm now rounds an optimal fractional solution $(x, y)$ of $(LP_B)$ to an integer solution $(\hat{x}, y')$, such that the total cost of the solution is no more than $4B$ and $\sum_i y'_i \leq 4k$. The algorithm begins with the process of *filtering*, where a new fractional solution $(\overline{x}, \overline{y})$ which costs not much more than $(x, y)$ but has the property that $\overline{x}_{ij}$ is positive only for $(i, j)$ which are close together. This allows us to re-assign vertices to roots as we round up $\overline{y}$ to an integer solution and set some $\overline{y}$ variables to zero. The process and a proof of its performance guarantee are described in detail in the next lemma.

**Lemma 6.** *Suppose that ($LP_B$) has a solution $(x, y)$ such that $\sum_{i \in V} y_i \leq k$. Then we can find in polynomial time a $4k$-star cover of cost at most $4B$.*

*Proof.* Define the fractional assignment cost $c_j$ as $c_j = \sum_{i \in V} x_{ij} c_{ij}$ for each $j \in V$. Now, define

$$\bar{x}_{ij} = \begin{cases} \min\{1, 2x_{ij}\} & : \text{ if } \quad c_{ij} \leq 2c_j \\ 0 & : \text{ otherwise} \end{cases}$$

and $\bar{y}_i = 2 \cdot y_i$ for all $i, j \in V$. It is not hard to see that $(\bar{x}, \bar{y})$ is a feasible solution for ($LP_B$), and $\sum_{i \in V} \overline{y}_i \leq 2k$. By scaling $y$, we can assume w.l.o.g. that $\sum_{i \in V} \bar{y}_i = 2k$.

We now define 0-1 variables $\hat{x}_{ij}$ as follows. Let $C$ denote the set of "unassigned" nodes and $R$ the set of "opened" roots. We initialize $C \leftarrow V$ and $R \leftarrow \emptyset$. As long as $C$ is non-empty, pick $v \in C$ that attains the minimum $c_v$ value among all nodes remaining in $C$, i.e. pick $v$ such that $c_v = \min_{u \in C} c_u$.

We add $v$ to $R$. For node $v$, let $\mathcal{F}_v$ be the set of roots that $v$ connects to, i.e. $\mathcal{F}_v = \{i \in V : \bar{x}_{iv} > 0\}$ and let $\mathcal{N}_v$ be the set of nodes in $C$ that are served by roots in $\mathcal{F}_v$, that is, $\mathcal{N}_v = \{j \in C : \exists i \in \mathcal{F}_v \text{ s.t. } \bar{x}_{ij} > 0\}$. We now assign all nodes in $\mathcal{N}_v$ to $v$, i.e. for all $j \in \mathcal{N}_v$, let

$$\hat{x}_{lj} = \begin{cases} 1 & : \text{ if } \quad l = v \\ 0 & : \text{ otherwise.} \end{cases}$$

Finally, we remove $\mathcal{N}_v$ from $C$ and continue, until $C = \emptyset$.

For every root $i \in R$, let $\hat{y}_i = \sum_{\{j : \hat{x}_{ij} = 1\}} 2c_j / B$. If $i \notin R$, then set $\hat{y}_i = 0$. We now claim that: (i) $\sum_i \hat{y}_i \leq 2k$. (ii) The cost of the star rooted at $v \in R$ is bounded by $2B \cdot \hat{y}_v$.

Firstly, (i) follows because of the following inequalities:

$$\sum_{i \in R} \hat{y}_i = \sum_{i \in R} \sum_{\{j : \hat{x}_{ij} = 1\}} \frac{2c_j}{B} \leq \frac{2}{B} \cdot B \cdot \sum_i y_i \leq 2k.$$

Now, observe that the reason for assigning node $j$ to the root $v$ must have been a (fractional solution) root $i$ that serves both $v$ and $j$, i.e. $\bar{x}_{ij} > 0$ and $\bar{x}_{iv} > 0$. But this means that $c_{jv} \leq c_{ij} + c_{iv} \leq 2c_j + 2c_v \leq 4c_j$ where the last inequality follows from our choice of $v$.

This proves (ii), since the cost of the star rooted at $v$ satisfies:

$$\sum_{\{j : \hat{x}_{vj} = 1\}} c_{jv} \leq \sum_{\{j : \hat{x}_{vj} = 1\}} 4c_j = 2B \cdot \hat{y}_v.$$

We now show that $|R| \leq 2k$. Consider any two vertices $u, v \in R$. By definition of our procedure of adding vertices to $R$, we must have $\mathcal{F}_u \cap \mathcal{F}_v = \emptyset$. However,

$\sum_i \bar{x}_{iv} \geq 1$ means $\sum_{i \in \mathcal{F}_v} \bar{y}_i \geq 1$ for all $v \in R$. Since we also have $\sum_{i \in V} \bar{y}_i \leq 2k$, we must have $|R| \leq 2k$.

Finally, we round the $\hat{y}_i$ variables to integer variables $y'_i$, defined as $y'_i = \lceil \hat{y}_i \rceil$. Since we have $|R| \leq 2k$ and $\sum_i \hat{y}_i \leq 2k$, we obtain $\sum_i y'_i \leq 4k$. We open $y'_i$ star centers at node $i$, thus creating at most $4k$ stars in total.

Using the fact that in $(IP_B)$ we only had $x_{ij} > 0$ for those $(i, j)$ pairs where $c_{ij} \leq B$, it follows that whenever $y'_i > 1$, we can assign the nodes served by $i$ to distinct stars centered at node $i$ such that each star has total cost no more than $4B$. This is because (i) we must have $c_v \leq B$ for all $v \in V$, meaning that $\hat{x}_{iv} > 0$ only if $c_{iv} \leq 2B$, and (ii) we have already guaranteed that the cost of the star rooted at $v$ in the solution given by $\hat{y}$ is bounded above by $2B \cdot \hat{y}_v$.

Thus we finish with an integral solution $y'$ and an integral assignment of nodes to stars given by $\hat{x}$, such that there are no more than $4k$ stars in total and each star has cost no more than $4B$.

Lemmas 5 and 6 yield the following theorem. This can also be converted into a strongly polynomial algorithm by applying the methods used in Section 4.1.

**Theorem 5.** *There is a polynomial-time algorithm for the un-rooted $k$-star cover problem that partitions the set of nodes into $4k$ stars each of which has cost bounded by $4B$, where $B$ is the value of an optimum solution.*

### 5.2 Rooted $k$-star cover

In the rooted version of $k$-star cover we are given a root set $R$ of cardinality $k$ in addition to the usual problem parameters and we are supposed to use the roots in $R$. Notice that this problem is equivalent to the following scheduling problem: We are given $k$ machines $M_1, \ldots, M_k$ (one for each root in $R$) and $n$ jobs $\{J_v\}_{v \in V}$. For each job–machine pair $(M_i, J_j)$ we have a processing time $c_{ij}$. The objective now is to assign each job to a unique machine. Let $J_i$ be the jobs assigned to machine $M_i$. We want to minimize the make-span, i.e. $\max_{1 \leq i \leq k} \sum_{j \in J_i} c_{ij}$. It is easy to see that this problem is equivalent to the rooted $k$-star packing problem. A 2-approximation due to Shmoys and Tardos [15] implies the following theorem:

**Theorem 6.** *There is a polynomial-time 2-approximation for the rooted $k$-star cover problem.*

## 6 Open questions

The more obvious integer programming formulation for $k$-star cover would minimize $B$ subject to opening no more than $k$ roots. However, we were unable to prove constant factor upper bounds on the integrality gap of that formulation. It would be interesting to see if that formulation, or some other technique, yields a constant factor approximation algorithm for $k$-star cover which obeys the constraint that no more than $k$ stars are used exactly.

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.

## Acknowledgments

We would like to thank Asaf Levin for sending us a copy of [1] and for his improved bi-criteria approximation algorithm for the unrooted $k$-star problem.

## References

1. E. Arkin, R. Hassin and A. Levin. Approximations for minimum and min-max vehicle routing problems. *Manuscript*, 2003.
2. Y. Bartal, M. Charikar and D. Raz. Approximating min-sum $k$-clustering in metric spaces. In *Proceedings of the* $33^{rd}$ *Annual ACM Symposium on Theory of Computing*, 11-20, 2001.
3. M. Charikar and R. Panigrahy. Clustering to minimize the sum of cluster diameters. In *Proceedings of the* $33^{rd}$ *Annual ACM Symposium on Theory of Computing*, 1-10, 2001.
4. F. Chudak and D. Shmoys. Improved approximation algorithms for a capacitated facility location problem. In *Proceedings of the* $10^{th}$ *Annual ACM-SIAM Symposium on Discrete Algorithms*, 875-876, 1999.
5. R. Diestel. *Graph Theory*, Springer-Verlag, Berlin, 2000.
6. M. Dyer and A. Frieze. A simple heuristic for the $p$-center problem. *Operations Research Letters*, 3(6):285-288, 1985.
7. 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.
8. G.N. Frederickson, M.S. Hecht and C.E. Kim. Approximation algorithms for some routing problems. *SIAM J. Computing* 7:178-193, 1978.
9. M. Garey and D. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W.H. Freeman, San Francisco, 1979.
10. N. Guttman-Beck and R. Hassin. Approximation algorithms for min-sum $p$-clustering. *Discrete Applied Mathematics*, 89:125-142, 1998.
11. M. Haimovich, A. Rinnooy Kan and L. Stougie. *Vehicle Routing: Methods and Studies*, Elsevier, 1988.
12. D. Hochbaum and D. Shmoys. A best possible approximation algorithm for the $k$-center problem. *Mathematics of Operations Research*, 10(2):180-184, 1985.
13. A. Levin, private communication, May 2003.
14. D. Shmoys, E. Tardos and K. Aardal. Approximation algorithms for facility location problems. In *Proceedings of the* $29^{th}$ *Annual ACM Symposium on Theory of Computing*, 265-274, 1997.
15. D. Shmoys and E. Tardos. An approximation algorithm for the generalized assignment problem. *Mathematical Programming A*, 62:461-474, 1993.
16. P. Toth and D. Vigo (editors). *The Vehicle Routing Problem*, SIAM monographs on discrete mathematics and applications, 2002.