A Constant-factor Approximation Algorithm for the k-MST Problem

(Extended Abstract)

Avrim Blum[*] R. Ravi[†] Santosh Vempala[‡]

Abstract

Given an undirected graph with non-negative edge costs and an integer k, the k-MST problem is that of finding a tree of minimum cost on k nodes. This problem is known to be NP-hard. We present a simple approximation algorithm that finds a solution whose cost is less than 17 times the cost of the optimum. This improves upon previous performance ratios for this problem – $O(\sqrt{k})$ due to Ravi et al., $O(\log^2 k)$ due to Awerbuch et al, and the previous best bound of $O(\log k)$ due to Rajagopalan and Vazirani. Given any $0 < \alpha < 1$, we first present a bicriteria approximation algorithm that outputs a tree on $p \geq \alpha k$ vertices of total cost at most $\frac{2pL}{(1-\alpha)k}$, where L is the cost of the optimal k-MST. The running time of the algorithm is $O(n^2 \log^2 n)$ on an n-node graph. We then show how to use this algorithm to derive a constant factor approximation algorithm for the k-MST problem. The main subroutine in our algorithm is identical to an approximation algorithm of Goemans and Williamsom for the prize-collecting Steiner tree problem.

[*]School of Computer Science, Carnegie Mellon University, Pittsburgh PA 15213. Supported in part by NSF National Young Investigator grant CCR-9357793 and a Sloan Foundation Research Fellowship. Email: avrim@cs.cmu.edu.

[†]Graduate School of Industrial Administration, Carnegie Mellon University, Pittsburgh PA 15213. Email: ravi+@cmu.edu.

[‡]School of Computer Science, Carnegie Mellon University, Pittsburgh PA 15213. Email: svempala@cs.cmu.edu.

STOC'96, Philadelphia PA,USA
© 1996 ACM 0-89791-785-5/96/05..$3.50

1 Introduction

Given an undirected graph $G = (V, E)$ with non-negative edge costs and an integer k, the k-MST problem is that of finding a tree of minimum cost that spans k vertices of G. We refer to a tree that spans k vertices as a k-tree. Note that we may assume that the edge costs satisfy the triangle inequality without loss of generality [11].

The main result of this paper is a constant factor approximation algorithm for the k-MST problem. This algorithm naturally extends to give constant factor approximations for several problems whose solution is based on the k-MST. One example is the quota-driven TSP in which we are given an undirected graph with distances on the edges and values (positive real numbers) on the vertices. Our goal is to find a tour such that the sum of the values of the vertices reached is at least some specified quota, while minimizing the total distance traveled. Other examples are the more general prize-collecting traveling salesman problem of Balas [2], and the orienteering problem of Golden, Levy and Vohra [8]. More details of the relation of these problems to the k-MST problem can be found in [1].

Previous work

The k-MST problem was shown to be NP-hard by R. Ravi, Sundaram, Marathe, Rosenkrantz, and S. S. Ravi [11] and independently by Zelikovsky and Lozevanu [12]. The former paper also presented an $O(\sqrt{k})$-approximation algorithm for this problem. This was improved by Awerbuch, Azar, Blum and Vempala [1] who gave an $O(\log^2 k)$-approximation algorithm. Recently, Rajagopalan and Vazirani [10] obtained an $O(\log k)$-approximation algorithm for the k-MST problem.

For the k-MST problem arising from points in the plane, Ravi et al. [11] presented an $O(k^{\frac{1}{4}})$-approximation algorithm. The approximation ratio was improved to $O(\log k)$ by Garg and Hochbaum [5], and subsequently to a constant factor by Blum, Chalasani and Vempala [4]. A smaller constant was obtained by Mitchell [9].

Main result

The rooted version of the k-MST problem requires inclusion of a specific root node in the k-tree. As observed in [1], solving the rooted and unrooted versions are essentially equivalent. We present a solution to the rooted version of the problem for simplicity.

Theorem 1 *There is an approximation algorithm for the rooted k-MST problem on general graphs with performance ratio at most 17 and running time $O(n^2 \log^4 n)$ on an n-node graph.*

The key ingredient in proving the above theorem is the following bicriteria approximation algorithm.

Theorem 2 *Given any $0 < \alpha < 1$, there is an approximation algorithm for the rooted k-MST problem that outputs a tree on $p \geq \alpha k$ vertices of total cost at most $\frac{2pL}{(1-\alpha)k}$, where L is the cost of the (optimal) k-MST. The running time of the algorithm is $O(n^2 \log^2 n)$ on an n-node graph.*

The algorithm used to prove the above theorem is identical to an approximation algorithm used by Goemans and Williamson [7] for the prize-collecting Steiner tree problem, with a particular setting of node penalties. We use key properties of their algorithm to show the performance ratio.

We can solve the unrooted problem by trying all the different vertices as roots and outputting the minimum tree obtained. This gives us an extra factor of n in the running time. As an easy consequence of Theorem 2 and Lemma 3 (see Section 4) we also get the following result for the unrooted problem.

Theorem 3 *Given any $0 < \alpha < 1$ there is an approximation algorithm for the unrooted k-MST problem that outputs a tree on p vertices, $\alpha k \leq p \leq 2\alpha k$, of cost at most $\frac{2pL}{(1-\alpha)k}$. The running time of the the algorithm is $O(n^3 \log^2 n)$.*

In the next section, we present the main bicriteria approximation algorithm used to prove Theorem 2. In the following section, we present the analysis of the performance ratio and the running time. In Section 4, we show how Theorem 1 follows from Theorem 2.

2 Algorithm

We will consider the rooted version of the k-MST problem where we are given a root r and the tree is required to contain the root. Let the cost of the optimal tree with k vertices be L. In this section we show how to find a tree with p vertices, $p \geq \alpha k$, of cost at most $p \cdot \frac{2L}{(1-\alpha)k}$ for any $0 < \alpha < 1$.

Our algorithm for the k-MST problem is identical to the approximation algorithm used by Goemans and Williamson for the prize-collecting Steiner tree problem. The prize-collecting Steiner tree problem is defined on an undirected graph with costs on edges, a subset of nodes specified as *terminals*, and nonnegative penalty values on the terminals. The goal is to find a tree such that the total cost of edges in the tree plus the penalties of all the terminals *not* in the tree is minimized.

2.1 Overview

In the version of the Goemans-Williamson algorithm we use, all nodes are designated terminals and have the same penalty value $\pi = \frac{L}{(1-\alpha)k}$.

We begin with an intuitive description. The penalty value π of a node can be thought of as the potential or charge assigned to it. The potential specifies how much "time" the node v can stay active. The algorithm begins with nodes in singleton clusters, each of radius 0 and with potential π.

The algorithm is perhaps better visualized as running in continuous, rather than discrete time. As time progresses, every cluster grows a breadth-first region around it, with all clusters growing at the same rate. To grow for a "width" or breadth-first distance of ϵ, the cluster must expend potential equal to ϵ. As the algorithm proceeds, some clusters may meet; for instance, the very first meeting will occur when the clusters growing from the two nearest neighbors in the graph meet at the midpoint of the edge between them. When two clusters meet, they are merged into a single cluster and their remaining potentials are added together to become the remaining potential of the new cluster. Another event that may happen is that a cluster may expend all its potential without meeting another cluster. In this case, the cluster stops growing and is deactivated. When a cluster is deactivated, the nodes inside are stamped with the "time of death" (technically, they are labeled with the set of vertices in the cluster).

A deactivated cluster in some ways is much like an active cluster: if an active cluster meets a deactivated one, the two will merge and have their potentials added together in the same way as done when two active clusters meet. On the other hand, once a vertex becomes labeled, it remains so forever.

The key property of this growing scheme is that the unlabeled nodes in a growing cluster can be connected together in a tree, which may also contain some of the labeled nodes of the cluster, such that the cost of the tree is not more than twice the potential π times the total number of nodes in the tree. Thus, this tree has an appropriate cost-to-nodes ratio as compared to an optimal k-tree. The growing scheme also allows us to argue that the algorithm will find such a tree of reasonably large size, thereby giving the result in Theorem 2.

The algorithm described below implements the above ideas in two phases. In the first phase we grow clusters, and in the second we prune inessential edges to create the desired tree. The reason for labeling vertices when their cluster becomes deactivated is to ensure that the tree produced in the second phase has sufficient potential to cover its cost. In particular, when we connect an unlabeled vertex to the tree, if the connection passes through a vertex with label C, then all other vertices with label C, or even with label $\hat{C} \supseteq C$, are placed into the tree as well. It can then be proven that this preserves the desired ratio.

One small modification to the above description is that for the sake of simplicity in the arguments, we consider the cluster containing the root node to be inactive. Whenever an active cluster merges with the root cluster, the resulting cluster becomes inactive even though the nodes in the cluster may have remaining potential to grow.

2.2 Description

The complete description of the algorithm is in Figure 1. The input to the algorithm is an undirected graph $G = (V, E)$ with edge costs $c_e \geq 0$, a root r, the cost L of an optimal k-tree containing r, and a fraction α. The algorithm outputs a tree F' containing r and at least αk nodes.

The algorithm runs in two phases. In the first we pick up edges and in the second we may delete some of the edges chosen in the first phase. In the first phase the algorithm maintains a forest F of edges. Initially the potential of each vertex, π_v, is set to $\frac{L}{(1-\alpha)k}$, where α is our input parameter. F is empty and hence each vertex is in a connected component by itself. All initial components except the one containing the root node are considered *active*. We can think of the potential of a node as the price it is willing to pay for connecting into a k-tree containing r.

At each step the algorithm does one of two things. First, it may add an edge between two connected components; if the resulting component contains the root it becomes inactive, otherwise it is considered active. Second, it may "deactivate" a component. Intuitively, a component is active if it is still growing and it is deactivated when we are unwilling to pay any more for vertices of this component. When a component is deactivated, the algorithm labels each unlabeled vertex within it (every node is initially unlabeled) with the set of all nodes in the component. The first phase ends when the component containing the root acquires more than $\alpha \cdot k$ unlabeled vertices. We argue in the next section that if the value of L is chosen correctly, this stopping condition is always reached.

In the second phase, we remove as many edges as we can from F while maintaining two properties: first, all unlabeled vertices in the root component must remain connected to the root node. Second, if a vertex with label C is connected to the root, then every vertex with label $\hat{C} \supseteq C$ must be connected to the root as well. Let C' be the resulting connected component containing the root, and F' the set of its edges.

To determine the choices in the first phase, the algorithm keeps a set of *growth* variables, y_S, one for each subset S of vertices. These are all initially implicitly set to zero. A growth variable can be positive for a subset of vertices, iff the vertices form an active component at some point in the first phase. (A useful property to keep in mind is that if $y_S > 0$ and $y_{S'} > 0$ then either S and S' are disjoint or else one of the two contains the other.) At each step of the first phase we increase uniformly the y_S's for the active components by a value ϵ which is the largest possible without violating one of the following constraints. For a subset of vertices S let $\delta(S)$ denote the set of edges with one end point in S and the other outside S.

a) For all $e \in E$,

$$\sum_{S: e \in \delta(S)} y_S \le c_e.$$

b) For all $T \subset V$,

$$\sum_{S \subseteq T} y_S \le \sum_{i \in T} \pi_i.$$

Increasing y_S's causes one of the above constraints (called "packing" constraints in [7]) to become tight. If a constraint of the first type becomes tight, that happens for some edge e between two connected components and the algorithm adds this edge to F. If a constraint of the second type becomes tight, this happens for some active component, and we then deactivate the component.

```
1   $F \leftarrow \emptyset$.
2   Comment: Implicitly set growth variables $y_S \leftarrow 0$ and cumulative growth variables $w(S) \leftarrow 0$ for all $S \subset V$. $w(S)$ denotes the total potential expended in component $S$ including any sub-components that were merged in creating $S$. Also implicitly set $\pi_v \leftarrow \frac{L}{(1-\alpha)k}$ for all nodes $v \neq r$.
3   $\mathcal{C} \leftarrow \{\{v\} : v \in V\}$.
4   For each $v \in V$ set $d(v) \leftarrow 0$.
    Comment: $d(v)$ denotes the distance of node $v$ to the boundary of the component containing $v$.
5   For each $v \in V$, if $v = r$ then $\lambda(\{v\}) \leftarrow 0$ else $\lambda(\{v\}) \leftarrow 1$.
    Comment: $\lambda(S)$ equals 1 if $S$ is active and 0 otherwise.
6   While the component $C_r$ containing the root has less than $\alpha k$ unlabeled nodes
7     Comment: Find the next event.
8     Find edge $e = (i,j)$ with $i \in C_p \in \mathcal{C}$, $j \in C_q \in \mathcal{C}$, $C_p \neq C_q$ that minimizes $\epsilon_1 = \frac{c_e - d(i) - d(j)}{\lambda(C_p) + \lambda(C_q)}$.
9     Find $\tilde{C} \in \mathcal{C}$ with $\lambda(\tilde{C}) = 1$ that minimizes $\epsilon_2 = |\tilde{C}| \cdot \frac{L}{(1-\alpha)k} - w(\tilde{C})$.
10    $\epsilon = \min(\epsilon_1, \epsilon_2)$.
11    $w(C) \leftarrow w(C) + \epsilon \cdot \lambda(C)$ for all $C \in \mathcal{C}$.
12    For all $v \in C_v \in \mathcal{C}$
13      $d(v) \leftarrow d(v) + \epsilon \cdot \lambda(C_v)$
14    If $\epsilon = \epsilon_2$   (i.e., $\tilde{C}$ is deactivated before $C_p$ and $C_q$ meet)
15      $\lambda(\tilde{C}) \leftarrow 0$.
16      Mark all unlabeled vertices of $\tilde{C}$ with label $\tilde{C}$.
    else   (i.e., $C_p$ and $C_q$ meet before $\tilde{C}$ is deactivated)
17      $F \leftarrow F \cup \{e\}$.
18      $\mathcal{C} \leftarrow \mathcal{C} \cup \{C_p \cup C_q\} - \{C_p\} - \{C_q\}$.
19      $w(C_p \cup C_q) \leftarrow w(C_p) + w(C_q)$.
20      If $r \in C_p \cup C_q$ then $\lambda(C_p \cup C_q) \leftarrow 0$ else $\lambda(C_p \cup C_q) \leftarrow 1$.
21  For every unlabeled vertex $v \notin C_r$
22    Mark $v$ with the label $C_v$ where $v \in C_v$.
23  $F'$ is derived from $F$ by removing as many edges as possible so that the following two properties hold: (1) every unlabeled vertex is connected to $r$; (2) if a vertex $v$ with label $C$ is connected to $r$, then so is every vertex with label $\hat{C} \supseteq C$.
24  $C'$ is the set of vertices spanned by $F'$.
```

Figure 1: The bicriteria approximation algorithm for k-MST.

3 Analysis

3.1 Upper bound

In the analysis of their algorithm for the prize-collecting Steiner tree problem, Goemans and Williamson use linear programming duality to compare the cost of their solution to an optimal one. Although we don't use an LP, we retain the basic elements of their analysis. A result equivalent to the following has been proven independently in [6].

Theorem 4 *The algorithm produces a connected component C' with a set of edges F' such that*

$$\sum_{e\in F'} c_e \leq 2\sum_{i\in C'} \pi_i.$$

Proof. By the construction, every vertex of C_r not spanned by F' lies in a component deactivated at some point in the algorithm. Further if a vertex v in a deactivated component C_i is not spanned by F' then no vertex of C_i is spanned by F'. With these observations we can partition the vertices of C_r not spanned by F' into disjoint deactivated components $C_1,..C_l$. Thus C_r is the disjoint union of the sets $C', C_1, \ldots C_l$.

Now consider the component C'. We must show that

$$\sum_{e\in F'} c_e \leq 2\sum_{i\in C'} \pi_i.$$

Since we have for each edge in F',

$$c_e = \sum_{S:e\in\delta(S)} y_S$$

and also that

$$\sum_{S\subseteq C'} y_S \leq \sum_{i\in C'} \pi_i,$$

it suffices to show that

$$\sum_{e\in F'}\sum_{S:e\in\delta(S)} y_S \leq 2\sum_{S\subseteq C'} y_S.$$

Adding in the growth values accumulated by deactivated components within C_r to both sides of the above equation, we need to show that

$$\sum_{e\in F'}\sum_{S:e\in\delta(S)} y_S + 2\sum_j\sum_{S\subseteq C_j} y_S \leq 2\sum_{S\subseteq C'} y_S + 2\sum_j\sum_{S\subseteq C_j} y_S.$$

Since C_r is the disjoint union of $C', C_1 \ldots C_l$, we have that $\sum_{S\subseteq C'} y_S + \sum_j \sum_{S\subseteq C_j} y_S = \sum_{S\subseteq C_r} y_S$. So we need to show that

$$\sum_{e\in F'}\sum_{S:e\in\delta(S)} y_S + 2\sum_j\sum_{S\subset C_j} y_S \leq 2\sum_{S\subseteq C_r} y_S$$

Rewriting the first term in the LHS (summing over S instead of over e), we finally get

$$\sum_{S\subseteq C_r} y_S|F'\bigcap\delta(S)| + 2\sum_j\sum_{S\subseteq C_j} y_S \leq 2\sum_{S\subseteq C_r} y_S. \qquad (1)$$

We show by induction that the above condition is maintained at every step of the algorithm. Initially this is true since all the y_S's are zero.

At some stage let $\mathcal{C}$ be the set of active components. Let H be the graph formed by considering active and inactive components that are subsets of C_r as vertices and the edges $e \in F'\bigcap\delta(C)$ for active $C \subseteq C_r$ as the edges of H. Discard vertices corresponding to isolated inactive vertices.

We need some more notation. Let N_a and N_i denote the active and inactive vertices in H respectively. Let N_d denote the vertices active in H, but in some inactive component C_j at the end of the algorithm. Finally, let deg_v denote the degree of a vertex in H. Note that N_d corresponds to vertices that are subsets of some deactivated component not spanned by F', so $N_d = \{v \in N_a : deg_v = 0\}$.

At the current step, the increase in the LHS of (1) is $\epsilon(\sum_{v\in N_a} deg_v + 2|N_d|)$ while the increase in the RHS is $2\epsilon|N_a|$. We would like to show that $\sum_{v\in N_a} deg_v + 2|N_d| \leq 2|N_a|$. Since the degree of a vertex in N_d is zero, it is enough to show that $\sum_{v\in N_a - N_d} deg_v + 2|N_d| \leq 2|N_a|$ which is equivalent to

$$\sum_{v\in N_a - N_d} deg_v \leq 2|N_a - N_d|.$$

To prove this we shall need one last fact, namely that all but one of the leaves of H are all active vertices. For suppose that v is an inactive leaf of H not containing r, adjacent to edge e, and let C_v be the connected component corresponding to v. Since C_v was deactivated, no vertex of C_v is unlabeled; Also since it is a leaf it is not on a path between any unlabeled vertex and r. So the edge e can be deleted in the second phase and $e \notin F'$, a contradiction.

$$\begin{aligned}\sum_{v\in N_a - N_d} deg_v &\leq \sum_{v\in(N_a-N_d)\bigcup N_i} deg_v - \sum_{v\in N_i} deg_v \\ &\leq 2(|(N_a - N_d)\cup N_i| - 1) - (2|N_i| - 1) \\ &\leq 2|N_a - N_d| - 1.\end{aligned}$$

We used above the fact that all but one inactive vertex have degree at least 2, and that H is a tree on the vertices $(N_a - N_d)\bigcup N_i$. ■

3.2 Termination and lower bound

Theorem 4 argued that the tree produced by the algorithm has the correct "ratio cost," namely the correct cost of the tree per node. To complete the proof of Theorem 2, we need to argue that the tree produced has the claimed number of nodes. Note that this condition is also required to ensure the stopping condition in step 6 of the algorithm. We do this next.

Suppose the value of L is chosen to be at least as much as the cost of a k-MST containing r. We then show that the root component will always acquire at least αk unlabeled vertices. We do this by showing that at most $(1-\alpha)k$ nodes of some optimal k-tree will belong to deactivated sets (i.e., be labeled) during the course of the algorithm. Before we do this, we need a couple of preliminaries.

Lemma 1 *Let S^* be a subset of nodes of a deactivated component C. The sum of growth values y_S assigned by the algorithm to subsets S such that $S \cap S^* \neq \emptyset$ is at least $|S^*|\cdot\pi = \frac{|S^*|\cdot L}{(1-\alpha)k}$.*

Proof. At any point in the running of the algorithm before C was deactivated, the nodes in C are partitioned into active and deactivated clusters. Let $C'_1 \dots C'_p$ denote the maximal subsets of C that were active at some point during the algorithm such that $C'_i \cap S^* = \emptyset$ for all i. Note that the sets C'_i are disjoint and that

$$\begin{aligned} |C| \cdot \pi &= w(C) = \sum_{S \subseteq C} y_S \\ &= \sum_{C'_i} \sum_{S \subseteq C'_i} y_S + \sum_{S \subseteq C, S \not\subseteq C'_i \text{ for any } i} y_S. \end{aligned}$$

The sets C'_i either became deactivated or merged with other active components that contained a node of S^*. Hence we have

$$\sum_{C'_i} \sum_{S \subseteq C'_i} y_S \leq \sum_i |C'_i| \cdot \pi$$

from the condition for deactivation. Thus we finally have

$$\begin{aligned} \sum_{S: S \cap S^* \neq \emptyset} y_S &= \sum_{S \subseteq C, S \not\subseteq C'_i \text{ for any } i} y_S \\ &= \sum_{S \subseteq C} y_S - \sum_{C'_i} \sum_{S \subseteq C'_i} y_S \\ &\geq |C| \cdot \pi - \sum_i |C'_i| \cdot \pi \\ &\geq |S^*| \pi. \end{aligned}$$

The last inequality follows since no C'_i contains any node of S^* by definition. ■

Lemma 2 *Let T^* be a tree spanning the root r and the nodes S^*. Let $\sum_{S \cap S^* \neq \emptyset} y_S = L'$. Then the cost of T^* is at least L'.*

Proof. Note that if $y_S > 0$ then $r \notin S$. A growth value of y_S assigned to the set S by the algorithm identifies a breadth-first cut of distance y_S around the set S. If $S \cap S^* \neq \emptyset$, then since T^* connects the nodes in S^* to the root r, it must cross this cut for at least distance y_S. Since the growth values assigned by the algorithm obey the packing constraints (a), the breadth-first distances identified by different cuts are disjoint. Hence the tree T^* must have cost at least $\sum_{S \cap S^* \neq \emptyset} y_S$. ■

Theorem 5 *Let T^* be a k-MST containing r of cost L. At most $(1-\alpha)k$ nodes of T^* are in deactivated sets (i.e., labeled) during the course of the algorithm.*

Proof. The proof is by contradiction. Let the node set of T^* be $S^* \cup \{r\}$. Suppose more than $(1-\alpha)k$ nodes of S^* are in deactivated components. By Lemma 1 for each deactivated component C_i,

$$\sum_{S: S \cap S^* \neq \emptyset,\ S \subseteq C_i} y_S \geq |C_i \cap S^*| \cdot \pi.$$

Summing over all deactivated components, we get

$$\begin{aligned} \sum_i \sum_{S. S \cap S^* \neq \emptyset,\ S \subseteq C_i} y_S &\geq \sum_i |C_i \cap S^*| \cdot \pi \\ &> (1-\alpha)k \cdot \pi = L. \end{aligned}$$

But then lemma 2 implies that the cost of T^* is greater than L, a contradiction. ■

3.3 Turning the proof into an algorithm

The algorithm in the proof of Theorem 2 assumes that L, the cost of a k-MST is known. One simple way to fix this lack of information is to run the algorithm for a guess value of L and perform binary search on the guess value depending on the outcome of the algorithm (a smaller value results in the algorithm terminating with fewer unlabeled nodes in the root component). This would require $O(\log \hat{L})$ invocations of the basic algorithm where $\hat{L}$ is the sum of the $k-1$ largest edge-costs in the graph.

The number of invocations of the basic algorithm can be reduced to $O(\log k)$ by providing an upper bound and a lower bound on the value of L that differ by a factor of at most k. Let ℓ denote the shortest distance such that there exists at least k nodes within distance ℓ from the root r. Then $\ell \leq L \leq k \cdot \ell$, and we have the required bound.

The running time of the algorithm then follows from noting that the basic algorithm can be implemented in $O(n^2 \log n)$ time using ideas from [7].

4 Completion

The algorithm presented so far has the following guarantee. Given an integer k, a bound L on the cost of the optimal k-MST, and $\alpha \in (0,1)$, the algorithm finds a tree on $p \geq \alpha k$ vertices of cost at most $p \cdot \frac{2L}{(1-\alpha)k}$.

There are two issues that must be dealt with to yield our final k-MST result. First, it is possible that the algorithm finds a tree with too many vertices; i.e., p is much larger than k. Second, if $p < k$ then we need to "boost" the tree found to a k-MST.

We handle the first problem as follows. Before running the algorithm, we remove all vertices of distance greater than L from the root, as these cannot possibly be in the optimal tree. We now run the algorithm. If the result is a tree on $p > k$ vertices, we apply the following lemma with $q = k$.

Lemma 3 *Given a tree T on p vertices and an integer $q \leq p$, we can find a subtree T' of T on p' vertices such that $p' \in [q, 2q]$ and $cost(T') \leq \frac{p'}{p} cost(T)$. The running time of this procedure is $O(n^2)$.*

Lemma 3 (with $q = k$) guarantees that the resulting tree T' has at least k vertices and cost at most $\frac{4L}{1-\alpha}$. We then pay an additional cost at most L to connect T' to the root, resulting in a total cost at most $L + \frac{4L}{1-\alpha}$. So, for instance, if we run the bicriteria algorithm with $\alpha = 1/2$ and it produces a tree on too many vertices, we can use this Lemma to find a k-tree of cost at most $9L$.

Proof of Lemma 3. If $p \leq 2q$ we are done. Otherwise, notice that in any tree of p vertices, there exists some vertex v such that removing v produces a forest in which each tree has at most $p/2$ vertices. Let $T_1, \ldots, T_d$ be the trees produced by removing v. Let p_i be the number of vertices in T_i and let C_i be the cost of T_i plus the length of the edge connecting T_i to v in the original tree. This means that the cost of T is $C_1 + \ldots + C_d$ and $p = p_1 + \ldots + p_d + 1$. Therefore, there must exist some i such that $C_i/p_i \geq cost(T)/p$. So, we simply remove tree T_i from T, which preserves (or improves) the cost-to-vertices ratio of the tree remaining and repeat. Notice that each iteration reduces the size of T by less than a factor of 2, so we can be assured that its size will eventually fall within our desired window. ■

We now handle the second problem listed: that of boosting the tree found in the case that it is too small. We do this using the notion of an (a, b)-tree approximator following [3]. An (a, b)-tree approximator is given quantities ϵ and L and has the following guarantee: if there exists a rooted tree on at least $(1-\epsilon)n$ vertices having total weight at most L, the algorithm will *find* a rooted tree on at least $(1 - a\epsilon)$ vertices having total weight at most bL. It is easy to see (as noted in [3]) that the results of Goemans and Williamson on approximating the prize-collecting Steiner tree problem [7] yield a $(3, 6)$-tree approximator. Goemans and Kleinberg [6] show that the Goemans-Williamson algorithm in fact produces a $(2, 4)$-tree approximator. Using this fact, we prove the following theorem:

Theorem 6 *Let L be an upper bound on the cost of the optimal rooted k-MST, and let $\gamma \in [0, 1/2]$. Given a rooted tree on $(1 - \gamma)k$ vertices having cost at most $4L$, in time $O(n^2 \log^2 n)$ we can produce either:*

(i) A rooted tree on at least $(1 - \frac{20}{21}\gamma)k$ vertices of cost at most $4L$, or

(ii) A rooted tree on at least k vertices of cost at most $17L$.

We can satisfy the preconditions of Theorem 6, in particular that $\gamma \leq 1/2$, by initially running the bicriteria algorithm with $\alpha = 1/2$. Assuming the "first problem" discussed above does not occur, this will find a tree on at least $k/2$ vertices with cost at most $4L$. (If the "first problem" *does* occur, then as noted above we can find a tree on k vertices of total cost at most $9L$ and we are done.) Now, applying Theorem 6 $O(\log k)$ times yields a constant factor solution to the k-MST. Note that we do this for each of possibly $O(\log k)$ guess values for L. This gives the performance ratio and the running time claimed in Theorem 1.

Proof of Theorem 6. The idea is similar to that used in [1] to reduce their performance ratio by a logarithmic factor. We are given a tree with $(1 - \gamma)k$ vertices. We know that in the remaining graph there exists a rooted tree on γk vertices of total cost at most L. We now apply our bicriteria approximation algorithm with $\alpha = \frac{5}{7}$ on the remaining graph. Let us assume for now that the tree returned has at most γk vertices, and so its cost is at most $7L$; we will return to the case that it has too many vertices at the end of the proof. (If the tree found has more than γk vertices, we can immediately achieve using Lemma 3 a k-tree of cost of at most $4L + (L + \frac{4L}{1-\alpha}) = 19L$: the extra complication is just in reducing this cost to $17L$.)

Let T be the union of our original tree and the new tree found, and p be the number of vertices in T. Note that $p \geq (1 - \gamma)k + \frac{5}{7}\gamma k = (1 - \frac{2}{7}\gamma)k$.

Define $\epsilon = \gamma/3$, and let us run the $(2, 4)$-tree approximator on the subgraph induced by the nodes of the tree T using this ϵ. If it is the case that the optimal tree has at least $(1 - \epsilon)p$ vertices inside T, then this approximator will find a tree on at least $(1 - 2\epsilon)p$ vertices of total cost at most $4L$. Using our definition of ϵ and our bound on p, this tree contains at least $(1 - \frac{2}{3}\gamma)(1 - \frac{2}{7}\gamma)k \geq (1 - \frac{20}{21}\gamma)k$ vertices, satisfying property (i) of the Theorem as desired.

If the approximator fails to find the desired number of vertices at the desired cost, it means that the optimal tree has fewer than $(1 - \epsilon)p$ vertices inside T, and therefore at least $k' = k - (1 - \epsilon)p$ vertices outside T. We now run our bicriteria algorithm one final time, with $\alpha = \frac{1}{2}$, on the remaining graph with tree T contracted to a root node. We are now guaranteed that our total number of vertices found is at least

$$\begin{aligned} p + \tfrac{1}{2}(k - (1 - \epsilon)p) &= \tfrac{1}{2}k + (\tfrac{1}{2} + \epsilon/2)p \\ &\geq \tfrac{1}{2}k + (\tfrac{1}{2} + \gamma/6)(1 - \tfrac{2}{7}\gamma)k \\ &= k + (\gamma/42 - \gamma^2/21)k \\ &\geq k. \quad (\text{since } \gamma \leq 1/2) \end{aligned}$$

If we did not run into our "first problem" of finding too many vertices in this run of the algorithm (i.e., we found between $\frac{1}{2}k'$ and k' vertices), we are done with total cost at most $4L + 7L + 4L = 15L$. If the tree found *did* have more than k' vertices, we apply Lemma 3 with $q = \frac{1}{2}k'$ to find a low cost subtree having between $\frac{1}{2}k'$ and k' vertices. In this case we may need to pay an additional cost L to connect the subtree to the root, for a total of $16L$.

We have now proven the theorem assuming that we are satisfied with a total cost of $19L$. To reduce the constant to 17 we must handle the case that when we ran the bicriteria algorithm with $\alpha = 5/7$, we found too many vertices. We do this by applying the algorithm of Lemma 3 with $q = \frac{5}{7}\gamma k$, and consider two cases depending on the number of vertices p' in the subtree found.

1. The first case is that $p' \in [\gamma k, \frac{10}{7}\gamma k]$. This means that the cost of the subtree is at most $\frac{10}{7} \cdot \frac{2L}{1-\alpha} = 10L$, or a total of $11L$ when we connect it to the root. Adding this cost to the $4L$ cost of our initial tree results in a k-tree of cost at most $15L$.

2. The second case is that $p' \in [\frac{5}{7}\gamma k, \gamma k]$. This means that the cost of the subtree is at most $7L$, or $8L$ when we connect it to the root. We can thus continue in the proof as if this were the tree returned by the bicriteria algorithm, paying an extra cost of L for a total of $17L$.

To show the running time, note that we used at most two calls to our bicriteria approximator, one call to the $(2, 4)$-tree approximator of [6], and one call to the procedure in the proof of Lemma 3. The tree approximator in [6] can be implemented using at most $\log n$ calls to the prize-collecting Steiner tree approximation algorithm of [7] giving a running time $O(n^2 \log^2 n)$. The bicriteria approximation has running time from Theorem 2. The procedure in Lemma 3 takes $O(n^2)$ time. Thus the overall running time is as claimed.

■

Acknowledgements. The second author thanks Naveen Garg and Balaji Raghavachari for stimulating discussions on the k-MST problem.

References

[1] B. Awerbuch, Y. Azar, A. Blum, and S. Vempala. Improved approximation guarantees for minimum-weight k-trees and prize-collecting salesmen. In *Proceedings of the 27th Annual ACM Symposium on Theory of Computing*, pages 277–283, May 1995.

[2] E. Balas. The prize collecting traveling salesman problem. *Networks*, 19:621–636, 1989.

[3] A. Blum, P. Chalasani, D. Coppersmith, B. Pulleyblank, P. Raghavan, and M. Sudan. The minimum latency problem. In *Proceedings of the 26th Annual ACM Symposium on Theory of Computing*, pages 163–171, 1994.

[4] A. Blum, P. Chalasani, and S. Vempala. A constant-factor approximation for the k-mst problem in the plane. In *Proceedings of the 27th Annual ACM Symposium on Theory of Computing*, pages 294–302, May 1995.

[5] N. Garg and D. Hochbaum. An $o(\log k)$ approximation algorithm for the k minimum spanning tree problem in the plane. In *Proceedings of the 26th Annual ACM Symposium on Theory of Computing*, pages 432–438, 1994.

[6] M. Goemans and J. Kleinberg. An improved approximation ratio for the minimum latency problem. In *Proceedings of the 7th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 152–158, 1996.

[7] M. Goemans and D. Williamson. A general approximation technique for constrained forest problems. *SIAM J. Computing* 24, pages 296–317, 995.

[8] B.L. Golden, L. Levy, and R. Vohra. The orienteering problem. *Naval Research Logistics*, 34:307–318, 1987.

[9] J.S.B. Mitchell. Guillotine subdivisions approximate polygonal subdivisions: A simple new method for the geometric k-MST problem. In *Proceedings of the 7th Annual ACM-SIAM Symposium on Discrete Algorithms*, 1996. to appear.

[10] S. Rajagopalan and V. Vazirani. Logarithmic approximation of minimum weight k trees. Unpublished manuscript, August 1995.

[11] R. Ravi, R. Sundaram, M.V. Marathe, D.J. Rosenkrantz, and S.S. Ravi. Spanning trees short and small. In *Proceedings of the 5th Annual ACM-SIAM Symposium on Discrete Algorithms*, 1994.

[12] A. Zelikovsky and D. Lozevanu. Minimal and bounded trees. In *Tezele Cong. XVIII Acad. Romano-Americane, Kishinev*, pages 25–26, 1993.