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

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 an approximation algorithm of Goemans and Williamsom for the prize-collecting Steiner tree problem.

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

*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. Supported in part by NSF CAREER grant CCR-9625297. Email: ravi@cmu.edu.

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

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 [15].

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 [3], and the orienteering problem of Golden, Levy and Vohra [11]. More details of the relation of these problems to the k-MST problem can be found in [2].

Previous work

The k-MST problem was shown to be NP-hard by R. Ravi, Sundaram, Marathe, Rosenkrantz, and S. S. Ravi [15] and independently by Fischetti et al. [6], and also by Zelikovsky and Lozevanu [16]. Ravi et al. [15] also presented an $O(\sqrt{k})$-approximation algorithm for this problem. This was improved by Awerbuch, Azar, Blum and Vempala [2] who gave an $O(\log^2 k)$-approximation algorithm. Recently, Rajagopalan and Vazirani [14] 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. [15] presented an $O(k^{\frac{1}{4}})$-approximation algorithm. The approximation ratio was improved to $O(\log k)$ by Garg and Hochbaum [8], and subsequently to a constant factor by Blum, Chalasani and Vempala [5]. A smaller constant was obtained by Mitchell [12].

Main result

The rooted version of the k-MST problem requires inclusion of a specific root node in the k-tree. As observed in [2], 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.*

Our proof of the above theorem involves a reduction to a prize-collecting Steiner tree problem. This 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.

We prove Theorem 2 by reducing our bicriteria problem to a version of a prize-collecting Steiner tree problem and applying an approximation algorithm of Goemans and Williamson [10] for the prize-collecting Steiner tree problem. However, the performance guarantee they prove for their algorithm is not sufficient for our purposes, and we prove a strengthening of it in Theorem 4.

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 1 (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 by showing a reduction to the prize-collecting problem. For the sake of completeness, we include a description of the Goemans-Williamson algorithm for prize-collecting Steiner trees. In the following section, we present our 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. Suppose we know that the cost of the optimal tree with k vertices is L. We will clarify this issue later in Section 3.1. 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 $0 < \alpha < 1$.

First, we reduce our problem to an instance of the rooted prize-collecting Steiner tree problem. The root node for the Steiner tree problem is the same as that for the k-MST problem. All nodes in the graph are designated terminal nodes and assigned the same penalty value of $\pi = \frac{L}{(1-\alpha)k}$. Next, we apply the approximation algorithm of Goemans and Williamson [10] to this prize-collecting problem. We include an intuitive overview and formal description of the prize-collecting approximation algorithm of Goemans and Williamson as applied to our problem for completeness[1]. In the next section, we prove a slight strengthening of their performance guarantee and use that in deriving our performance bounds in Theorem 2.

2.1 Overview

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. Viewed as a rooted prize-collecting Steiner tree problem, the algorithm tries to find a tree containing the root r such that the total cost of the edges in the tree plus the sum of the penalties of nodes not in the tree is minimized. Note that the penalty term is exactly equal to the number of nodes not in the tree times the uniform penalty value of $\frac{L}{(1-\alpha)k}$.

The Goemans-Williamson algorithm for this problem is based on the primal-dual method applied to the following linear programming relaxation of the prize-collecting problem. For a subset of vertices S let $\delta(S)$ denote the set of edges with one endpoint in S and the other outside S.

$$
\begin{array}{llcll}
\text{Minimize} & \displaystyle\sum_{e \in E} c_e x_e + \sum_{v \neq r} (1 - z_v)\pi_v & & & \\
\text{subject to} & \sum_{e \in \delta(S)} x_e & \geq & z_v & v \in S; r \notin S \\
 & x_e & \geq & 0 & e \in E \\
 & z_v & \geq & 0 & v \in V
\end{array}
$$

In the integral strengthening of the program, the variables x_e and z_v are required to be binary. The indicator variable z_v denotes whether the node v is included in the tree, and the nontrivial constraints insist that cuts which separate the root from a node included in the tree must be covered or crossed at least once. The dual to the above linear programming relaxation is the following.

$$
\begin{array}{llcll}
\text{Maximize} & \displaystyle\sum_{S : r \notin S} y_S & & & \\
\text{subject to} & \sum_{S : e \in \delta(S)} y_S & \leq & c_e & e \in E \\
 & \sum_{S \subset T} y_S & \leq & \sum_{v \in T} \pi_v & T \subset V; r \notin T \\
 & y_S & \geq & 0 & S \subset V; r \notin S
\end{array}
$$

We begin with an intuitive description of the algorithm. The algorithm runs in two phases. In the first phase we grow clusters while building a tree for each

[1] Readers familiar with this algorithm can skim the overview and description of the algorithm.

cluster hand in hand, and in the second we prune inessential edges to retain the desired tree.

In the first phase the algorithm grows clusters while maintaining a forest F, of edges that contains one tree per cluster. Each cluster also has a potential value which is used to guide the growth process. We can think of the potential of a cluster as the price it is willing to pay for connecting into the tree containing r. Initially, F is empty and hence each vertex is in a connected component (cluster) by itself. All initial components except the one containing the root node are considered *active*, i.e., ready to grow. The potential of every initial cluster (node) is set to $\pi_v = \frac{L}{(1-\alpha)k}$, where α is our input parameter.

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. At this point, the edge along which they meet is added to connect up the trees of the two merging clusters into a single tree for 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).

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 that results 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.

The values assigned to the growth vairables by the algorithm are such that they form a feasible solution to the dual program for the prize-collecting problem. In this way, the two events mentioned above correspond to making the two different types of constraints in the dual program tight. More formally, at each step of the first phase we increase uniformly the y_S's for all the active components by a value ϵ which is the largest possible without violating one of the following two constraints arising from the dual program:

a) For all $e \in E$,

$$\sum_{S: e \in \delta(S)} y_S \leq c_e. \tag{1}$$

b) For all $T \subset V$,

$$\sum_{S \subseteq T} y_S \quad \leq \quad \sum_{i \in T} \pi_i. \tag{2}$$

Increasing y_S's causes one of the above constraints 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. In this way, we maintain that the set of variables y_S form a feasible solution to the dual of the prize-collecting problem. Suppose OPT' denotes an optimal solution for the prize-collecting problem. We get the following observation as a consequence of weak duality.

Fact 1

$$OPT' \geq \sum_S y_S$$

In the second phase, we prune some edges from the forest F found in the first phase to obtain the final solution. In particular, 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.

A simple two-step procedure can accomplish the pruning in the second phase: first we consider the subtree formed by all edges in F that lie on some path between an unlabeled node and the root. Then, for every labeled vertex in this subtree, if its label is C, we retain all edges that connect vertices with labels $\hat{C} \supseteq C$ to the root. Note that if a tree in F does not contain the root node, then all its edges are deleted.

The complete description of the algorithm is in Figure 1.

3 Analysis

Our analysis relies on the following observation which was made independently in [9]. Let OPT' be the cost of a minimum solution to the prize-collecting Steiner tree problem.

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

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

$F \leftarrow \emptyset$.

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 $v \neq r$.

$\mathcal{C} \leftarrow \{\{v\} \ : \ v \in V, v \neq r\}$.

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.

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.

While there are active components

    Comment: Find the next event.

    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 $\epsilon_1 = \frac{c_e - d(i) - d(j)}{\lambda(C_p) + \lambda(C_q)}$.

    Find $\tilde{C} \in \mathcal{C}$ with $\lambda(\tilde{C}) = 1$ that minimizes $\epsilon_2 = \left|\tilde{C}\right| \cdot \frac{L}{(1-\alpha)k} - w(\tilde{C})$.

    $\epsilon = \min(\epsilon_1, \epsilon_2)$.

    $w(C) \leftarrow w(C) + \epsilon \cdot \lambda(C)$ for all $C \in \mathcal{C}$.

    For all $v \in C_v \in \mathcal{C}$

        $d(v) \leftarrow d(v) + \epsilon \cdot \lambda(C_v)$

    If $\epsilon = \epsilon_2$ (i.e., $\tilde{C}$ is deactivated before C_p and C_q meet)

        $\lambda(\tilde{C}) \leftarrow 0$.

        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)

        $F \leftarrow F \cup \{e\}$.

        $\mathcal{C} \leftarrow \mathcal{C} \cup \{C_p \cup C_q\} - \{C_p\} - \{C_q\}$.

        $w(C_p \cup C_q) \leftarrow w(C_p) + w(C_q)$.

        If $r \in C_p \cup C_q$ then $\lambda(C_p \cup C_q) \leftarrow 0$ else $\lambda(C_p \cup C_q) \leftarrow 1$.

For every unlabeled vertex $v \notin C_r$

    Mark v with the label C_v where $v \in C_v$.

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 $\hat{C} \supseteq C$.

C' is the set of vertices spanned by F'.

Figure 1: The approximation algorithm for prize-collecting trees applied to find a bicriteria k-MST.

The theorem says that the sum of the costs of the edges of the tree output by the algorithm plus twice the penalties on nodes not in the tree is at most twice the minimum possible. We now exploit this fact via an appropriate choice of penalties on the nodes, $\pi_i = \frac{L}{(1-\alpha)k}$.

Corollary 5 *The cost of the tree output by the algorithm is at most $\frac{2pL}{(1-\alpha)k}$ and it has at least αk vertices.*

Proof. Let $\pi_i = \frac{L}{(1-\alpha)k}$. Let p be the number of vertices in the final tree output. Then from Theorem 4 we have

$$\sum_{e \in F'} c_e + 2(n-p).\frac{L}{(1-\alpha)k} \leq 2(L + (n-k)\frac{L}{(1-\alpha)k}).$$

Rearranging terms,

$$\sum_{e \in F'} c_e \leq \frac{2(p-\alpha k)L}{(1-\alpha)k}$$

whence the claim. ∎

Proof of Theorem 4. The proof of this theorem essentially follows from the proof of the analogous theorem in [10] by bounding only the cost of the tree constructed by the algorithm. We reproduce it here for completeness.

Recall that C_r is the component containing the root at the termination of the first phase of the algorithm. After the pruning in the second phase, a subset of the nodes $C' \subseteq C_r$ are retained in the final tree F'.

First, we consider vertices that are not in the component C_r. Since all such vertices belong to deactivated components (at termination) and we maintain the condition in Equation 2, we have that

$$\sum_{i \notin C_r} \pi_i = \sum_{S : S \subseteq V \setminus C_r} y_S \tag{3}$$

Next consider the vertices in C_r. 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 $C', C_1, \ldots C_l$.

Using Fact 1 that $\sum_S y_S$ is a lower bound on OPT', what we need to show is

$$\sum_{e \in F'} c_e + 2\sum_j \sum_{i \in C_j} \pi_i + 2\sum_{i \notin C_r} \pi_i \leq 2 \sum_{S \subseteq V \setminus C_r} y_S + 2\sum_{S \subseteq C_r} y_S.$$

Using Equation 3 above, this reduces to showing

$$\sum_{e \in F'} c_e + 2\sum_j \sum_{i \in C_j} \pi_i \leq 2\sum_{S \subseteq C_r} y_S,$$

which is the same as

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

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

$$\sum_{S \subseteq C_r} y_S |F' \bigcap \delta(S)| + 2 \sum_j \sum_{S \subseteq C_j} y_S \leq 2 \sum_{S \subseteq C_r} y_S. \tag{4}$$

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 (4) 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. Therefore

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

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 $(N_a - N_d) \bigcup N_i$. ■

3.1 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 [10].

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 1 *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 1 (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 1. 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 [4]. 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 [4]) that the results of Goemans and Williamson on approximating the prize-collecting Steiner tree problem [10] yield a $(3, 6)$-tree approximator. Goemans and Kleinberg [9] show that the Goemans-Williamson algorithm in fact produces a $(2, 4)$-tree approximator. These approximators work by calling the prize-collecting Steiner tree algorithm with a suitable choice of penalties. 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 [2] to reduce their performance ratio by a logarithmic factor. We are given a rooted tree with $(1-\gamma)k$ vertices. We know that in the graph obtained by contracting the current tree nodes into the root, 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 1 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. By our metric assumption on the costs, note that the subgraph induced by T has distances equal to the shortest path distance in G. If it is the case that the optimal tree has at least $(1-\epsilon)p$ vertices inside T, then the 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 1 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 1 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 [9], and one call to the procedure in the proof of Lemma 1. The tree approximator in [9] can be implemented using at most $\log n$ calls to the prize-collecting Steiner tree approximation algorithm of [10] giving a running time $O(n^2 \log^2 n)$. The bicriteria approximation has running time from Theorem 2. The procedure in Lemma 1 takes $O(n^2)$ time. Thus the overall running time is as claimed.
■

5 Subsequent work

Subsequent to our work, Garg [7] has shown how a variant of our approach achieves a 3-approximation algorithm for the k-MST problem. He also shows that the integrality gap of a natural integer programming formulation for the k-MST problem is also three, which suggests that a different approach may be needed to further improve the performance ratio for the k-MST problem. For the case of the k-MST problem on points in the plane, polynomial-time approximation schemes have been derived independently by Arora [1] and Mitchell [13].

Acknowledgements. The authors thank Naveen Garg and Balaji Raghavachari for stimulating discussions on the k-MST problem.

References

[1] S. Arora. Plynomial-time approximation schemes for Euclidean TSP and other geometric problems. In *Proceedings of the 37th Annual Symposium on Foundations of Computer Science*, pages 2-13, 1996.

[2] 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.

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

[4] 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.

[5] 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.

[6] M. Fischetti, H.W. Hamacher, K. Jrnsten, F. Maffioli. Weighted k-cardinality trees:complexity and polyhedral structure. In *Networks*, 24, pages 11-21, 1994.

[7] N. Garg. A 3-Approximation for the Minimum Tree Spanning k Vertices. In *Proceedings of the 37th Annual Symposium on Foundations of Computer Science*, pages 302-309, 1996.

[8] 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.

[9] 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.

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

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

[12] 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*, pages 402-408, 1996.

[13] J.S.B. Mitchell. Guillotine subdivisions approximate polygonal subdivisions: Part II - A simple polynomial-time approximation scheme for geometric k-MST, TSP, and related problems. Manuscript, April 30, 1996. To appear, SIAM J. Comp.

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

[15] 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.

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