Improved Approximation Guarantees for Minimum-Weight k-Trees and Prize-Collecting Salesmen

Baruch Awerbuch * Yossi Azar † Avrim Blum ‡
Santosh Vempala‡

Abstract

We consider a formalization of the following problem. A salesperson must sell some quota of brushes in order to win a trip to Hawaii. This salesperson has a map (a weighted graph) in which each city has an attached demand specifying the number of brushes that can be sold in that city. What is the best route to take to sell the quota while traveling the least distance possible? Notice that unlike the standard traveling salesman problem, not only do we need to figure out the order in which to visit the cities, but we must decide the more fundamental question: which cities do we want to visit?

In this paper we give the first approximation algorithm having a poly-logarithmic performance guarantee for this problem, and approximate as well the slightly more general "Prize Collecting Traveling Salesman Problem" (PCTSP) of Balas, and a variation we call the "Bank-robber Problem" (also called the "Orienteering Problem" by Golden, Levi, and Vohra). We do this by providing an $O(\log^2 k)$ approximation to the somewhat cleaner k-MST problem which is defined as follows. Given an undirected graph on n nodes with non-negative edge weights and an integer $k \leq n$, find the tree of least weight that spans k vertices. (If desired, one may specify in the problem a "root vertex" that must be in the tree as well.) Our result improves on the previous best bound of $O(\sqrt{k})$ of Ravi et al.

Keywords: Approximation, Prize Collecting Traveling Salesman Problem, k-MST.

AMS subject classifications: 68Q20, 68Q25, 90C27, 90B06, 05C85

Running head: Approximation Guarantees for the k-MST and PCTSP

*Johns Hopkins University, Baltimore, MD 21218, and MIT Lab. for Computer Science. E-mail: baruch@blaze.cs.jhu.edu. Supported by Air Force Contract TNDGAFOSR-86-0078, ARPA/Army contract DABT63-93-C-0038, ARO contract DAAL03-86-K-0171, NSF contract 9114440-CCR, DARPA contract N00014-J-92-1799, and a special grant from IBM.

†Department of Computer Science, Tel Aviv University. E-Mail: azar@math.tau.ac.il. Research supported in part by Allon Fellowship and by the Israel Science Foundation administered by the Israel Academy of Sciences.

‡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. E-mail: {avrim,svempala}@cs.cmu.edu.

1 Introduction

The problem. Consider a salesperson that must sell some quota of R brushes in order to win a trip to Hawaii. This salesperson has a map (a weighted graph) of n cities in which each city has an attached demand specifying the number of brushes that can be sold in that city. What is the best route to take to sell the quota while traveling the least distance possible? Notice that unlike the standard traveling salesman problem, not only do we need to figure out the order in which to visit the cities, but we must decide the more fundamental question: which cities do we want to visit?

R. Ravi, Sundaram, Marathe, Rosenkrantz, and S.S. Ravi [RSM+94] considered the cleanest case of above problem, called the minimum-weight k-tree, or k-MST problem. In this problem, one is given a graph on n vertices with non-negative distances on the edges, and a number $k \leq n$, and the goal is to find a tree of least total cost that spans k vertices. For $k = n$ this is the (easy) minimum spanning tree problem. For general k, however, the problem is NP-complete and has the same main difficulty faced by the above salesperson: which points to include and which to ignore? In fact, the k-MST problem nicely focuses on just that issue since once the set is determined, the least weight tree on that set is easy to find.

Cheung and A.Kumar [CK94] call this problem the "quorum-cast" problem, whose application are in the domain of communication networks. Other applications for this problem include fault-tolerant distributed computing and data management.

The bank robber problem is the following: given the map of a city including the amounts of money in each bank, and a car with bounded gas tank, the robber has to rob the maximum amount of money without refueling after first robbery (thus avoiding being reported to the police). This problem is also called the "orienteering problem" by Golden, Levi, and Vohra [GLV87].

Existing work. Ravi et al. [RSM+94] provide an algorithm that achieves an approximation ratio of $O(\sqrt{k})$ for the k-MST problem on general graphs (i.e., the tree found is at most $O(\sqrt{k})$ times heavier than the optimal tree) and ratio $O(k^{1/4})$ for the special case of points in 2-dimensional Euclidean space. Garg and Hochbaum [GH94] improved the ratio for the latter case to $O(\log k)$, which has since been improved to a constant factor by Blum, Chalasani, and Vempala [BCV95].

Heuristics for problems described above have been given by Balas [Bal89] and by Cheung and A.Kumar [CK94].

Results of this paper. In this paper, we describe an algorithm that achieves an approximation ratio $O(\log^2 k)$ for the k-MST problem on general graphs, improving the bound of [RSM+94] and coming closer to the bounds for the special case of points in the Euclidean plane. Our results hold for both the rooted and unrooted versions (is there a required "start" vertex?). This result immediately implies an $O(\log^2 R)$ approximation for the quota-driven salesperson described above (R is the quota): namely, just treat a vertex with "demand" d as a cluster of d vertices, find the R-MST, and then tour the tree in the standard way. In fact, our algorithm actually achieves the somewhat better bound of $O(\log^2(\min(R,n)))$ for this problem, and does not require the demands to be polynomial in n.

Our algorithm also extends easily to a $O(\log^2(\min(R,n)))$ bound for the prize-collecting traveling salesman problem (PCTSP) due to Balas [Bal89] on undirected graphs. The PCTSP problem is just like the quota TSP problem but in addition there are non-negative penalties attached to each city and the salesperson's cost is the sum of the distance traveled plus the penalties on cities *not* visited. Thus, the quota problem can be thought of as the special case in which penalties are 0. The $O(\log^2(\min(R,n)))$ bound for the PCTSP follows immediately by concatenating the tour found by our algorithm (which ignores the penalties) to a tour found by a 2-approximation algorithm by Goemans and Williamson [GW92] to a relaxed version of the PCTSP in which the quota requirement is removed. (In the original PCTSP there is also a restriction that each city not be visited more than once; if this is desired, we can achieve the same bound in the standard way if the graph is a metric space.)

We also derive an approximation algorithm with similar bounds for the bank robber problem.

It is worthwhile to point out that our algorithms are easily implementable in distributed environment, since they operate on the basis of local information.

2 The k-MST problem

We begin by presenting an algorithm for the k-MST problem that achieves an approximation ratio of $O(\log^3 k)$. We then describe an improvement that removes one of the logarithmic factors to achieve the ratio of $O(\log^2 k)$. Before presenting the algorithm, however, let us point out that the "rooted" and "unrooted" versions of the problem are essentially equivalent from the point of view of approximation for the following reason.

Given an algorithm for the rooted problem, to solve the unrooted case one can simply try all possible start vertices and then choose the smallest tree found. Given an algorithm for the unrooted version, to solve the rooted case when the weight ℓ of the optimal tree is known just throw out all vertices of distance greater than ℓ from the root, solve the unrooted problem, and then connect the tree to the root for an added cost of at most ℓ. Note that the approximation factor may increase by 1. If the optimal cost ℓ is not known, simply sort the distances from the root to each of the n points in increasing order, run the algorithm n times throwing out the i farthest points in the ith iteration, and pick the best result.

In the rest of this section we will use OPT to denote the optimal k-tree and ℓ to denote its total weight.

Our algorithm and analysis contain two main ideas. The first is a measure used for grouping points into components in a Kruskal's-algorithm-like manner. The second is a bucketing technique that allows one to prove this measure to be useful. The measure we use is the following: given two components C_i, C_j, we examine the ratio: $d(C_i, C_j)/\min(|C_i|, |C_j|)$, where $d(\cdot,\cdot)$ is the distance according to the shortest-path metric and $|\cdot|$ is the size in terms of number of points. The general step of the algorithm will be joining together (using the shortest path) the two components for which this ratio is smallest.

The bulk of the argument will be for proving correctness of an algorithm for the following slight relaxation of our goal, which is similar to the "maximal dense" tree concept in [AAG93]. Given k, we will find a tree on at least $k/4$ points whose weight is at most $O(\log^2 k)$ times the weight of the minimum k-tree. With this algorithm in place, it will be easy to remove the relaxation and solve our original problem. The Kruskal-like algorithm for this relaxed problem is as follows:

Algorithm Merge-Cluster:

1. Begin with n components, one for each point.
2. Join (using the shortest path) the two components such that the ratio of the distance between the components to the number of points in the smaller one is least. That is, join the pair C_i, C_j that minimize $d(C_i, C_j)/\min(|C_i|, |C_j|)$.
3. Repeat Step (2) until some component has size at least $k/4$.

Theorem 1 *The weight of the largest component produced by Algorithm* **Merge-Cluster** *is at most* $4(\log_2 k)^2$ *times the weight of the optimal* k*-tree.*

The proof of Theorem 1 follows immediately from Lemmas 1 and 2 below.

Lemma 1 *If at any time the largest ratio used by algorithm* **Merge-Cluster** *so far is* r*, then any component of* p *points will have total weight at most* $rp\log_2 p$*.*

Lemma 2 *Algorithm* **Merge-Cluster** *never uses a ratio larger than* $(8\ell \log_2 k)/k$ *where* ℓ *is the weight of the optimal* k*-tree.*

To prove Theorem 1 from these lemmas, just note that the only way in which the largest component produced could have size greater than $k/2$ is for the additional vertices to be included "for free" in the shortest path that makes up the final connection. Thus combining the bounds of the two lemmas yields the theorem.

We begin with a proof of the simpler lemma.

Proof of Lemma 1. Consider a joining of two components. Since the length of the connection used is at most r times the number of points in the smaller component, we can "pay for" the connection by charging a cost of at most r to each of the points in the smaller component. Any time a point is charged, the size of the component it belongs to at least doubles. So, any point in a component of p points has been charged a total cost at most $r\log_2 p$. Since the weight of a component is at most the total charge to points inside it, this proves the lemma. ∎

Proof of Lemma 2. In contradiction, suppose at some time all components produced by the algorithm have size less than $k/4$ and the distance between any two is greater than $r = (8\ell\log_2 k)/k$ times the number of points in the smaller. Group the components into buckets based on size, where the ith bucket contains those components with between $k/2^i$

and $k/2^{i+1}$ points ($i = 2, 3, \ldots$). Now, throw out all components that do not intersect the optimal k-tree. Clearly the optimal k-tree can have at most $k/4 + k/8 + \ldots < k/2$ points inside buckets that contain only one component. So, there is some bucket containing at least 2 components such that OPT has at least $k/(2 \log_2 k)$ points inside that bucket. Say all components in this bucket have size between s and $2s$. This means that the balls of radius $rs/2$ about each component do not touch each other and OPT must intersect at least $k/(4s \log_2 k)$ components. Therefore OPT must have a connection cost greater than $rk/(8 \log_2 k) = \ell$, a contradiction. ■

Algorithm **Merge-Cluster** immediately gives us a simple $O(\log^3 k)$ approximation algorithm for the k-MST problem as follows. For simplicity, we consider the rooted version. Also, for the moment suppose that we know the weight ℓ of the optimal k-tree. In the procedure below, we view Algorithm **Merge-Cluster** as taking "k" as an argument.

Algorithm Connect-Clusters:

1. Mark as "to be ignored" all vertices of distance greater than ℓ from the root.
2. While $k > 0$ do the following:
   (a) Run algorithm **Merge-Cluster** on the unmarked vertices. (By this we mean that the distance between two components is still the shortest path distance in the original graph, but only unmarked vertices are considered in computing a component's size.)
   (b) Let s be the size of the component that was found and mark its vertices as "to be ignored".
   (c) Set $k = k - s$ (number of vertices we still need).
3. Connect together all the components found in Step 2.

Theorem 2 *Algorithm* **Connect-Clusters** *finds a tree of at least k points whose weight is at most $O(\log^3 k)$ times the optimal.*

Proof. Suppose in the invocations of Algorithm **Merge-Cluster** so far we have found components with k' points total. Then, the optimal k-tree contains at least $k - k'$ points in the graph remaining, and all these are within distance ℓ from the root. Thus, the next invocation of the algorithm will find a tree on at least $(k - k')/4$ points, at cost at most $O(\ell \log^2 k)$. So the algorithm will be run at most $O(\log k)$ times and the sum total cost of all components found is at most $O(\ell \log^3 k)$. The cost to connect them together is a low-order $O(\ell \log k)$. ■

We can remove the knowledge of the optimal cost ℓ from the above algorithm in the same manner as was done for converting the rooted version of the k-MST problem to the unrooted version. For improved efficiency, note that the true ℓ satisfies $\lambda \leq \ell \leq k\lambda$, where λ is the distance of the kth farthest vertex from the root. So we can begin with a guess of $\ell = \lambda$ and then double our guess if the numbers and sizes of the components found do not satisfy the guaranteed bounds, for a total of $O(\log k)$ iterations maximum.

We now show how to modify Algorithm **Connect-Clusters** to achieve an $O(\log^2 k)$ approximation. To do this, we use the following corollary to a result by Goemans and Williamson [GW92]. In [BCC+94] this is called a $(3,6)$-TSP approximator.

Fact 1 *Given a weighted graph on n points and an $\epsilon > 0$, let L_ϵ be the length of the shortest tour that visits at least $(1-\epsilon)n$ points. One can find in polynomial time a path of length at most $6L_\epsilon$ that visits at least $(1-3\epsilon)n$ points.*

For simplicity, we describe the modified algorithm as either finding a tree of k points with cost at most $O(\ell \log^2 k)$ or else finding a tree on at least $k/4$ points with cost $O(\ell)$. It is not hard to see that this suffices because the latter case removes an $O(\log^2 k)$ factor from the bounds of Theorem 1 (which is even better). The new algorithm works as follows.

Algorithm Improved-Connect: Run Algorithm **Connect-Clusters** until components totaling at least $\frac{15}{16}k$ points have been found. This requires only a constant number of applications of Algorithm **Merge-Cluster**. For simplicity, if the number of points found exceeds $\frac{15}{16}k$, then discard points until we have only $\frac{15}{16}k$ left. If the optimal k-tree intersects less than a $(1-\frac{3}{15})$ fraction of these points (and so contains at least $k/4$ new points), then one final application of **Merge-Cluster** (with argument $k/4$) will find a new component with at least $k/16$ points and we are done. On the other hand, if the optimal k-tree intersects at least a $(1-\frac{3}{15})$ fraction of these points, then by applying the algorithm of Fact 1, we can find a path of length $O(\ell)$ that visits at least $(1-\frac{9}{15})\frac{15}{16}k = \frac{3}{8}k$ points. Thus, the MST on these points is a tree of cost $O(\ell)$ on more than $k/4$ points.

We thus have the following theorem.

Theorem 3 *Algorithm* **Improved-Connect** *provides an $O(\log^2 k)$ approximation for the k-MST problem and runs in polynomial time.*

3 Extensions of the basic k-MST algorithm

We now describe how the algorithms of the previous section can be used to give guaranteed approximations to the other problems mentioned in the introduction, such as

- the quota TSP problem,
- the prize-collecting salesman problem, and
- the bank robber (orienteering) problem.

3.1 Algorithms for quota-driven salesmen

In the quota TSP problem each vertex in the graph has some attached integral value $w_i \geq 0$ and the salesman has a target quota R. The goal is to find a route as short as possible that visits vertices whose sum total value is at least R. The salesman may visit a given city more than once. (If salesman is restricted to one visit per city, the same approximation ratios can be reached in the standard way in the case of a complete graph where distances obey the triangle inequality.)

First, it is immediate that we can approximate the quota TSP to a factor of $O(\log^2 R)$. Simply replace each vertex of value w by w vertices all at the same location, find the approximate R-MST, and then traverse it at most twice. Notice that this bound might not be so good if R is much larger than n. (Also, this approach naively requires R to be only polynomially large; however, since the first step of the k-MST approximation algorithm is to reconnect vertices at the same location into a cluster, we can view the replacement described above as just a thought experiment.) We show now that the algorithm in fact achieves the better bound of $O(\log^2(\min(R, n)))$.

It will be simplest to view Algorithm **Merge-Cluster** as acting directly on the weighted vertices, merging the two components C_i, C_j that minimize $d(C_i, C_j)/\min(wt(C_i), wt(C_j))$ where $wt(C)$ is the sum of the values of the vertices contained in C. Let us call this algorithm **Merge-Weighted-Cluster** (even though it is really exactly the same algorithm, except for running time, as the thought experiment described above). For the analysis corresponding to Lemma 1, however, when two components are merged we will "pay for" the cost by charging to the smaller one in *number*, not in weight. This still means that for a connection of ratio r a vertex of weight w will be charged at most rw, if we charge vertices proportionally to their weight. But, now it is clear that a vertex will be charged at most $\log(p)$ times if it is in a component of p vertices, as opposed to a component having *weight* p. Thus we have the following lemma. (We also give a more formal proof below.)

Lemma 3 *If at any time, the largest ratio used by the algorithm* **Merge-Weighted-Cluster** *so far is r, then any component of p points and total vertex-weight w will have total edge weight (cost) at most $rw \log_2 p$.*

Proof: We prove it by induction. It is true initially since the cost begins at 0. When merging two clusters C_i and C_j into C we note that

$$\begin{aligned}
cost(C) &= cost(C_i) + cost(C_j) + d(C_i, C_j) \\
&\leq r \cdot wt(C_i) \cdot \log(|C_i|) + r \cdot wt(C_j) \cdot \\
&\quad \cdot \log(|C_j|) + r \cdot \min\{wt(C_i), wt(C_j)\} \\
&\leq r \cdot (wt(C_i) + wt(C_j)) \cdot \log(|C_i| + |C_j|) \\
&\leq r \cdot wt(C) \cdot \log(|C|) \quad \blacksquare
\end{aligned}$$

We can similarly improve Lemma 2 as follows.

Lemma 4 *Algorithm* **Merge-Weighted-Cluster** *never uses a ratio larger than* $O(\ell(\log_2 n)/R)$ *where* ℓ *is the (edge) weight of the optimal tree having vertex-weight* R.

Proof. Following the proof of Lemma 2 we have buckets containing the components of vertex-weight $R/4$ to $R/8$, $R/8$ to $R/16$, etc. We stop, however, at weight $R/(10n)$ and put all components of that weight or less into one single bucket. Now there are only $O(\log n)$ buckets instead of $O(\log R)$ and the final small bucket intersects the optimal tree in at most $R/10$ total weight and so can be "thrown out" in the analysis. The rest of the proof of Lemma 2 then can be followed directly. ■

The above two lemmas imply that Algorithm **Improved-Connect** of Theorem 3 in fact achieves a ratio of $O(\log^2 n)$ as well as $O(\log^2 R)$, which gives us our desired bound.

3.2 Algorithms for prize-collecting salesmen

As already mentioned in the introduction, an approximation algorithm to the quota TSP problem can be transformed into an approximation algorithm to the PCTSP problem [Bal89] (which has the additional complication of penalties attached to vertices and the "cost" of a tour equals its length plus the sum of penalties on points not visited) as follows. Concatenate the tour found by the quota TSP approximator to a tour found by a 2-approximation algorithm of [GW92] to a version of the PCTSP in which the quota requirement is removed. (Removing the quota restriction only decreases the cost of the optimum solution.) Thus we have the following theorem.

Theorem 4 *There is a polynomial time algorithm that approximates the PCTSP problem of [Bal89] on* n*-vertex undirected weighted graphs to a ratio* $O(\log^2(\min(R, n)))$*, where* R *is the required vertex weight to be visited.*

3.3 The bank robber (orienteering) algorithm

The bank robber (orienteering) problem [GLV87] is much like the problem faced by our quota-driven salesperson, except that the distance d that may be traveled is fixed and the goal is to maximize the total value R of points visited. If we do not require a specified starting point, then we can approximate this problem to the same ratio as the quota-TSP problem as follows. We "guess" the value R, we run the quota TSP approximator to find a path of length $O(d \log^2(\min(n, R)))$ visiting vertex-weight R, we break the path found into segments of length $d/2$, and then we choose the segment that contains the most vertex-value inside. Notice, however, that this does not approximate the orienteering problem with a specified start vertex (root) since there is no guarantee the "good" segment found will intersect the root.

4 A hard example for our algorithm

Our basic algorithm, **Merge-Cluster**, finds a tree on at least $k/4$ points with cost at most $O(\ell \log^2 k)$, where ℓ is the cost of the optimal k-tree. In fact, there exist examples that force the algorithm to pay $\Omega(\ell \log^2 k)$, and we describe one such example here.

Define a 1-block to be a single point, a 2-block to be two points separated by distance 1, a 4-block to be two 2-blocks separated by distance 2, and more generally a 2^t-block to be two 2^{t-1}-blocks separated by distance 2^{t-1}. Note that a 2^t-block has 2^t points. Also, notice that all the points in such a block would be connected together by **Merge-Cluster** using connections of ratio 1.

Suppose we have a cluster of A points (by a cluster we just mean that all the points are at the same location) separated by some distance from a cluster of B points. Let us define a "filling in" of the region between the two clusters as follows. First, if $A = 1$ or $B = 1$ then we do nothing. Otherwise, let $C = \lfloor \min(A/2, B/2) \rfloor$ and place a cluster of C points halfway between the A-point cluster and the B-point cluster. Then recursively fill in the region between the A-point and C-point clusters and the region between the C-point and B-point clusters.

A hard example for the algorithm can now be described as follows. The graph consists of two sets of points separated by a large distance. The first set of points is a k-block. The second set of points is constructed by placing two clusters of $4k/\log k$ points at a distance of $8k/\log k$ from each other, and then "filling in" the region between the clusters as described above. Given such a graph, there exists a tree on at least k points with total length $8k/\log k$. In particular, just connect all the points in the second set. However, the algorithm will instead connect together points in the first set, since the ratios are better there (1 versus 2), and pay a total cost of $\Omega(k \log k)$.

5 Open questions

The obvious open question is whether there exist polynomial-time algorithms with better approximation ratios, i.e. logarithmic, or even constant, for the problems considered in this paper. Alternatively, one would like to prove that improving the approximation is impossible unless P=NP.

Another interesting open question is finding a polynomial-time poly-logarithmic approximation for the rooted version of the bank robber (orienteering) problem. This problem appears to be much more difficult. Intuitively, the difficulty with approximating the rooted problem is that many of the points on the optimal tour might be in a clump at distance just about $d/2$ from the root. Thus, a strategy that opportunistically visits nearer vertices on its way to that clump may find that it cannot reach those vertices and return in the given distance d.

Acknowledgements

We thank Noga Alon and Prasad Chalasani for helpful discussions.

References

[AAG93] Baruch Awerbuch, Yossi Azar, and Rainer Gawlick. Dense trees and competitive selective multicast. unpublished manuscript, December 1993.

[Bal89] E. Balas. The prize collecting traveling salesman problem. *Networks*, 19:621–636, 1989.

[BCC+94] 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.

[BCV95] 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, 1995.

[CK94] Shun Yan Cheung and Akhil Kumar. Efficient quorumcast routing algorithms. In *Proceedings of INFOCOM '94*, volume 2, pages 840–855, Toronto, Ontario, 1994.

[GH94] N. Garg and D. Hochbaum. $O(\log k)$ 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.

[GLV87] B.L. Golden, L. Levy, and R. Vohra. The orienteering problem. *Naval Research Logistics*, 34:307–318, 1987.

[GW92] M. Goemans and D. Williamson. General approximation technique for constrained forest problems. In *Proceedings of the 3rd Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 307–315, 1992.

[RSM+94] 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.