The k-Traveling Repairmen Problem

JITTAT FAKCHAROENPHOL

Kasetsart University

CHRIS HARRELSON

Google

AND

SATISH RAO

UC Berkeley

40

Abstract. We consider the k-traveling repairmen problem, also known as the minimum latency problem, to multiple repairmen. We give a polynomial-time 8.497α-approximation algorithm for this generalization, where α denotes the best achievable approximation factor for the problem of finding the least-cost rooted tree spanning i vertices of a metric. For the latter problem, a $(2+\epsilon)$-approximation is known. Our results can be compared with the best-known approximation algorithm using similar techniques for the case $k = 1$, which is 3.59α. Moreover, recent work of Chaudry et al. [2003] shows how to remove the factor of α, thus improving all of these results by that factor. We are aware of no previous work on the approximability of the present problem. In addition, we give a simple proof of the 3.59α-approximation result that can be more easily extended to the case of multiple repairmen, and may be of independent interest.

Categories and Subject Descriptors: F.2.2 [**Analysis of Algorithms and Problem Complexity**]: Nonnumerical Algorithms and Problems—*Routing and layout, geometrical problems and computations*

General Terms: Algorithms, Theory

Additional Key Words and Phrases: Traveling salesman, vehicle routing

ACM Reference Format:

Fakcharoenphol, J., Harrelson, C. and Rao, S. 2007. The k-traveling repairmen problem. ACM Trans. Algor. 3, 4, Article 40 (November 2007), 16 pages. DOI = 10.1145/1290672.1290677 http://doi.acm.org/10.1145/1290672.1290677

Authors' addresses: J. Fakcharoenphol, Department of Computer Engineering, Kasetsart University; C. Harrelson, Google Inc.; S. Rao, University of California at Berkeley, Berkeley, CA.

1. *Introduction*

Informally, in the *k-traveling repairmen* problem we are given as input a set of k repairmen at a common depot s, and n customers sitting in some metric space at prescribed distances from each other and the depot. The goal is to find those tours on which to send the repairmen that minimize the average time a customer has to wait for a repairman to arrive, while making sure that all customers are served. A *(polynomial-time) approximation algorithm* for this problem with approximation factor c always returns a set of tours with average latency at most c times the average latency of the optimal solution.

Minimizing average wait time is a very natural objective. In addition to the namesake application of scheduling actual repairmen or deliverymen, consider the following situation. We are given a computer network, and told that a prize has been placed randomly on a node of the network. As pointed out by Blum et al. [1994], finding a set of tours that minimize the objective function of the k-traveling repairmen problem is the same as finding an optimal routing to minimize the expected time to find the prize with k "web crawlers."

2. *Related Work*

As far as we know, our work is the first to study the present problem and, in particular, the first to study it from the perspective of approximation algorithms. This is somewhat suprising, since it is a natural extension of the one-repairman case, and has practical applications.

However, after the publication of the conference version of our work, Chekuri and Kumar [2004], using a similar algorithm to ours, generalized our results to the case of multiple depots, vertex weights, and customer dwell times. They gave a 12α-approximation algorithm for these more general problems. As stated, their result is a 24α-approximation, but a simple optimization reduces this to 12α. This can most likely be improved further with the techniques in this article, but since their algorithm uses the same basic technique as ours, it is not likely to be improved beyond a bound on our algorithm.

2.1. ONE-REPAIRMAN VERSION.

The case where $k = 1$ has been studied extensively and also goes by several other names, including the "minimum latency problem," the "schoolbus driver problem," and the "deliveryman problem". It is known to be NP-hard for a general metric [Sahni and Gonzales 1976], and, as recently shown, is even NP-hard when the metric is known to be a weighted tree [Sitters 2002].

In 1994, the first constant-factor approximation algorithm was given by Blum et al. [1994], who gave an algorithm with an approximation factor of 144. They also showed how to produce an 8α-approximation, where α is the approximation factor of another problem, the *i-MST problem*. However, at the time no constant-factor approximation was known for the latter. Plugging in the currently best-known approximation factor for i-MST of $2 + \epsilon$ (for any $\epsilon > 0$) [Arora and Karakostas 2000], we obtain a $(16 + \epsilon)$-approximation. Goemans and Kleinberg subsequently showed how to improve the approximation factor to $\approx 3.59\alpha \leq 7.18 + \epsilon$ via a carefully designed refinement of the algorithm and analysis of Blum et al. [Goemans and Kleinberg 1996]. Archer, Levin, and Williamson [Archer et al. 2003; Archer

and Williamson 2003] recently gave a much faster algorithm that has an approximation factor of $\approx$7.18 (without the ϵ). Finally, Chaudry et al. [2003] noticed that one can replace the i-MST subroutine with what they call the *i-stroll problem*. They used this to remove the factor of α from all Blum et al.-like algorithms (including ours). We discuss i-MST and i-stroll in more detail in Section 3.

It is also known that the one-repairman case (and hence ours as well) is hard to approximate better than some constant factor [Blum et al. 1994]. On the other hand, Arora and Karakostas [1999] have also developed a pseudopolynomial-time approximation scheme for some geometric graphs and trees.

2.2. The Traveling Salesman Problem and Vehicle Routing. On the other hand, the traveling *salesman* problem (where the objective function is the total length of the tour, not the average latency) has been generalized to multiple salesmen. For this problem, we only allow each salesman to visit q cities (think of q as about n/k). This situation is a version of "vehicle routing," where one wants to schedule vehicles to carry objects from specified source vertices to specified destination ones. (The k-traveling salesman problem with capacity q can be seen as the task of using k capacity-q limited vehicles to transport n identical objects that start at s, one to each customer.)

Haimovich et al. [1988] showed that simply dividing a single-salesman tour piecewise among the salesmen achieves a constant-factor approximation for this problem. We note that this does not work for the k-repairmen problem; it is easy to see that such a solution may be a factor of $\Omega(k)$ away from optimal. It is also possible, and not too hard, to show that there is a simple tour-cutting method (similar to the technique of Haimovich et al. [1988]) that will result in tours of cost within a factor of $O(k)$ of the optimal k-repairmen solution. Since this only gives an $O(k)$, approximation to k-repairmen, we do not give the details. However, this shows in particular that the Haimovich et al. technique will not work well for our problem.

Some other related work includes that of Asano et al. [1997], who considered a similar vehicle routing problem in a geometric setting, attaining a PTAS for very large k. Christofides [1985] also gave a survey of various issues related to vehicle-routing variants of the traveling salesman problem. We also note that vehicle-routing algorithms are still very actively studied; in fact, a new book on the subject has recently been published [Toth and Vigo 2002].

3. *Definitions and Summary of Results*

We now make the definition of the problem more formal. In the *k-traveling repairmen problem*, we are given as input a metric d on a set of vertices V, and a special vertex $s \in V$ (which will be the starting point/depot). We use the terms "customer" and "vertex" interchangeably. The aim is to find k disjoint tours of the vertices, all of which start at s, that together cover all of the vertices in V, and minimize the sum of the latencies of tours.

The *cost* of a tour which visits vertices $x_0 = s, x_1, x_2, \ldots, x_m$, in that order, is $\sum_{i=0}^{m} d_i$, where $d_i = \sum_{j=0}^{i-1} d(x_j, x_{j+1})$ is the length of the tour up to vertex x_i. We also say that d_i is the *latency* of a vertex in the ith position of some tour that is implicitly under consideration. For a collection of disjoint tours, we define the cost of the collection to be sum of the costs of each individual tour.

Define the cost of a tree T on the vertices as the sum, over all edges (u, v) in T, of $d(u, v)$. For a tree T, an *E*uler tour of T is a tour over the vertices of T in any postorder traversal (see Cormen et al. [1990] for details).

Let M_k^* and MLT_k^* denote the optimal cost and optimal set of tours with k repairmen, respectively. In particular, M_1^* and MLT_1^* denote the optimum cost and tour with one repairman. Note that the cost of a tour can also be written in the (often more insightful) form $\sum_{i=0}^{m-1} d(x_i, x_{i+1})(m-i)$. The first definition emphasizes the amount of latency that a specific vertex *experiences*. The second emphasizes the amount of latency that a vertex *imposes* on the vertices subsequent to it in the tour by delaying them. in this article, we will take advantage of both formulations.

Next we define two problems which are used as subroutines. The first is the (rooted) i-MST *problem*. In that problem, we are given as input a metric d on a set of vertices V, and a special vertex $s \in V$ (which will be the root of the tree). An algorithm for this problem finds the least expensive tree which has exactly i vertices and includes the vertex s. We will repeatedly call this subroutine and use the trees returned as subtours in our solution. The i-MST problem is NP-hard, but a polynomial-time algorithm which outputs a tree with cost within a multiplicative factor of $2+\epsilon$ of optimal [Arora and Karakostas 2000] is known. A more practical 3-approximation algorithm is also known [Garg 1996].

The second subroutine is the (rooted) *i-stroll problem* [Chaudry et al. 2003] in which we are again given as input a metric d and special vertex $s \in V$. The difference is that now we would like to find the least expensive *path* that starts at s and contains i vertices. This problem is of course also NP-hard, with no better approximation algorithm known than for i-MST. However, Charikar et al. [2001] observed that there nevertheless exists a polynomial-time algorithm that outputs a *tree* with cost at most that of the optimal path solution. This can be exploited to improve our results as follows.

We now begin with a succinct statement of our main result. It is worth noting again that as far as we are aware, this is the first-known approximation algorithm for the k-traveling repairmen problem.

THEOREM 3.1. *Let α be the best polynomial-time approximation factor possible for the i-MST problem. Then there exists an* 8.497α *polynomial-time approximation algorithm for the k-traveling repairmen problem.*

The following improvement, due to Chaudry et al., is worth noting and the best known for this problem. Their innovation is based on the observation that the lower bound in observation 1) to follow can be improved to the optimal *i-stroll solution*, since the optimal tour is, of course, itself a path. They then show how to output a tree which has cost at most that of the optimal path; this improves the lower bound on the optimal solution by a factor of α.

THEOREM 3.2 [CHAUDRY ET AL. 2003]. *There exists an* 8.497 *polynomial-time approximation algorithm for the k-traveling repairmen problem.*

Our proofs proceed in the same spirit as those of Blum et al. [1994] for the $k = 1$ case. In order to explain our intuition, we first need to describe their approximation algorithm. It is based on the following two observations:

(1) The latency of a customer v at position i in the optimal tour MLT_1^* is lower-bounded by the cost of the i-MST solution. It's easy to see that this is true since,

after all, the part of MLT_1^* which starts at s and reaches v is a candidate to be the least expensive such tree.

(2) If we traverse subtours of exponentially increasing length, the latency of a customer that appears in the jth such subtour is dominated by the cost of the jth subtour (and not subtours 1 through $j-1$).

Given these observations, and assuming that we have available an optimal algorithm for the i-MST problem, here is the Blum et al. algorithm:

Algorithm. Blum et al.

For $j = 0, 1, \ldots$ find the tree, rooted at s, which spans the most vertices among those that have cost at most 2^{j+1}. Traverse a Euler tour of each of these trees in increasing order of j, short-circuiting tours to avoid going to vertices twice.

To analyze the quality of this algorithm, start with observation 1. Consider a vertex v in position i in the optimal tour MLT_1^*. It has latency in the range $[2^j, 2^{j+1})$ for some $j \geq 0$ (assume that all distances are at least 1). Hence, when we create the tree of cost at most 2^{j+1} in their algorithm, we span at least i vertices, since the optimal tour is 1 candidate. This means that the ith vertex in our tour (which need not be v) has latency at most

$$\sum_{l=0}^{j}(\text{cost of } l\text{th tour}) \leq \sum_{l=0}^{j} 2 \cdot 2^{l+1} \leq 2^{j+3}$$

(each term in the sum is 1 Euler tour of a tree with cost at most 2^{l+1}; also note that this sum is where observation 2 comes in, since the jth tour has cost $2 \cdot 2^{j+1} \approx 2^{j+3}$). Hence, their algorithm's ith vertex has latency at most a factor of $2^{j+3}/2^j = 8$ larger than v. Since this analysis works for all vertices, this algorithm gives an 8-approximation.

Of course, the problem of finding the biggest tree of cost at most 2^{j+1} is NP-hard. However, we could compute this tree if we could repeatedly compute the least expensive tree rooted at s with i vertices, increasing i until the cost exceeds 2^{j+1}. The latter problem is the aforementioned i-MST problem, which is also NP-hard. Luckily, however, using an α-approximation to the i-MST problem blows up the cost of the returned tree by only a factor of α. Therefore, using such an approximation algorithm yields a final approximation ratio of 8α for the k-traveling repairmen problem.

If we could find these trees via a similar method for all of our k repairmen, then we would obtain a constant-factor approximation to the k-traveling repairmen problem. More specifically, for any $j \geq 0$, we would like to be able to find a set of k trees, each with cost at most 2^{j+1}, which are disjoint except for s, and span the most vertices possible. However, it is not immediately clear how to find these trees. Lemma 5.1 shows that these trees would also give a factor 8 approximation if we could somehow calculate them.

In order to find these trees, we make two key additional observations:

(3) If we relax the requirement that we find *exactly* k trees, we can use a greedy, set cover-like approach to cover at least as many vertices with $k \ln n$ trees of cost at most 2^{j+1} each. In this scheme, each repairman traverses at most $\ln n$ trees of each cost. This leads to an $O(\log n)$-approximation (Lemma 5.3).

(4) If we relax the requirement to find the most possible number of vertices, but instead settle for a constant fraction of them, we can do it with only $O(k)$ trees. This leads to a constant-factor approximation algorithm (Lemma 6.3).

Next we show how to optimize the approximation factor. In Section 7 we give a simpler proof of the 3.59α-approximation improvement for the single-repairman case (Lemma 7.1), and extend its ideas to improve on the constant achieved by our algorithm that uses the aforesaid observation 2 (Lemma 7.5).

4. *Technical Issues*

Since multiplying all distances by a universal constant does not change which tour is optimal, we assume throughout that all distances are strictly larger than 2^{j_0} for some convenient constant j_0, which will turn out to depend on which algorithm we are running. For all but two of our algorithms, we use $j_0 = 0$, meaning that all distances are at least 1. Algorithm 4 requires $j_0 = 1$, and for Algorithm 5, the situation becomes slightly more complicated. To do this, we also assume that there are no vertices at distance zero from s, which is possible without loss of generality since we can, of course, always traverse such vertices first in the tour without increasing any latencies.

Second, although our algorithms iterate for $j = 0, 1, 2, \ldots$ without ever appearing to stop, in fact we implement them by calling an i-MST subroutine with only n possible different values of i. Even with the removal of vertices required by Algorithms 4 and 5, it is clear that there are only $O(n^2)$ different calls to the subroutine. Hence our algorithms can easily be made to run in polynomial time.

4.1. Reducing Budget to Cardinality.

Our algorithms frequently need to compute a rooted tree spanning the most vertices and having cost at most some budget B. We now show how to appproximately reduce this obstacle to the i-MST problem. For this purpose, let algorithm M be an α-approximation algorithm for i-MST.

Algorithm. Subroutine BudgetTree (B)

```
T = [empty tree]
for i = 1 to n do:
    if M(i) has cost at most αB then
        T = M(i)
return T
```

Lemma 4.1. *If there exists a tree of cost at most B that spans i vertices, then BudgetTree(B) always returns a tree of cost at most αB that spans at least i vertices.*

Proof. Suppose there exists a tree of cost at most B that spans i vertices. Then, by definition, $M(i)$ must return a tree of cost at most αB. □

Note that we do not care whether BudgetTree returns a tree if there is no tree within the budget.

5. *The O(*Log *N)-Approximation Algorithm*

First we show that the k-traveling repairmen problem is well approximated by a notion of minimum spanning trees similar to that of Blum et al. [1994]. Let $\text{MST}_k(C)$ denote a spanning tree of a subset of V, rooted at s, with out-degree at most k at s, for which each "subtree" (i.e., connected component after removing s, plus the edge to s) has cost at most C, and which spans the most total vertices.

Algorithm. 1

For $j = 1, 2, \ldots$ do:
    Find the tree $\text{MST}_k(2^{j+1})$.
Give each repairman one arbitrary subtree from the tree for each j, and tell him to traverse them in increasing order of j, short-circuiting as necessary to avoid going to vertices twice.

Note that this algorithm is not directly useful, since it solves an NP-hard problem as a subroutine.

LEMMA 5.1. *Algorithm* 1 *outputs a set of tours of cost at most* $8M_k^*$.

PROOF. Consider the optimal set of k tours. List all vertices in increasing order of their latencies within their tours. Let v_i be the ith vertex in this list. It has latency in the range $[2^j, 2^{j+1})$ for some j. The tree $\text{MST}_k(2^{j+1})$ must then cover at least i vertices, since the optimal set of k tours restricted to the first i vertices is one candidate if we regard all k subtours to be the relevant subtrees. The ith vertex in our sorted list of vertices has latency at most $\sum_{l=0}^{j} 2 \cdot 2^{l+1} \leq 2^{j+3} = 8 \cdot 2^j$, which is at most a factor of 8 larger than the latency of v_i.

Since this is true for all vertices v_i, we see that our tour has cost at most 8 times the cost of the optimal set of tours, which completes the lemma. □

Computing $\text{MST}_k(2^{j+1})$ exactly is NP-hard. We now give an algorithm which spans at least as many vertices, but potentially uses many more subtrees.

Algorithm. Subroutine 2

On input j:
For $l = 1, 2, \ldots, k \ln n$ do:
    Let $T_l = \text{BudgetTree}(2^{j+1})$. Remove all vertices of the tree T_l other than s from V.
Return the union of these trees.

LEMMA 5.2. *For every* j, *the* $k \ln n$ *trees* $\{T_l\}_l$ *found in Subroutine* 2 *cover at least as many vertices as* $MST_k(2^{j+1})$.

PROOF. The proof of this lemma uses essentially the same idea as the well-known $O(\log n)$-approximation algorithm for set cover.

Fix $j \geq 0$. Let $i_0 = i$ be the number of vertices other than s spanned by $\text{MST}_k(2^{j+1})$; by analogy with set cover, say also that they are *covered*. Consider some step l in the preceding algorithm. Let $i_l = i - \sum_{m=1}^{l} |T_m|$, for $l \geq 0$. (Here $|T_m|$ denotes the number of vertices in tour T_m but not in any earlier one.) Before the first step, there are i_0 vertices covered by $\text{MST}_k(2^{j+1})$ and not by our algorithm. If $i_l \leq 0$, we have covered at least i vertices already. If $i_l \leq k$, by Lemma 4.1 we will clearly be able to cover at least one more vertex in each remaining step until $i_l \leq 0$, at which point we are done. This will hence take at most k steps.

We now argue that after at most $k \ln n - k$ steps, we will be in one of the previous two cases, and hence finish covering vertices in at most $k \ln n$ steps. Assume without loss of generality that $i_0 > k$. We now show by induction on l that $i_l \leq i_0(1-1/k)^l$, and hence $i_{k \ln n - k} \leq k$. Clearly, if we have finished step l and $i_l > k$, there are at least i_l vertices in $\mathrm{MST}_k(2^{j+1})$ which have not yet been found in one of the previous trees. Then there must be some subtree T among the k in $\mathrm{MST}_k(2^{j+1})$ which contains at least $\lceil i_l/k \rceil \geq i_l/k$ vertices which have not yet been covered by our algorithm.

The most inexpensive tree containing all of these vertices has cost at most 2^{j+1}, since they constitute only a part of T, which has cost at most 2^{j+1}. Hence by Lemma 4.1 we can cover at least i_l/k vertices in step l. Hence $i_{l+1} \leq i_l - i_l/k = i_l(1-1/k)$. By transitivity, $i_l \leq i_0(1-1/k)^l$. Since $i_0 = i \leq n$, setting $l = k \ln n - k$ reduces this figure below k, which proves the lemma. □

Now we can write an algorithm which uses Subroutine 2 to approximate $MST_k(j)$.

Algorithm. 3

For $j = 1, 2, \ldots$ do
  Find the $k \ln n$ trees as in Subroutine 2 with parameter j, and
  assign $\ln n$ of the trees to each repairman arbitrarily.
Tell each repairman to traverse Euler tours of his trees in order of increasing j. For each j, he may traverse his corresponding set of at most $\ln n$ trees in an arbitrary order, short-circuiting to avoid going to vertices twice.

LEMMA 5.3. *This algorithm outputs a solution with cost at most* $8\alpha \ln n$ *times* M_k^*.

PROOF. Once again, sort all of the vertices in the optimal solution MLT_k^* globally by latency. Consider the vertex v with the ith smallest latency. It has latency in the range $[2^j, 2^{j+1})$ for some $j \geq 0$. Thus, at least i vertices are covered in Subroutine 2 with parameter j, by Lemma 5.2. Hence Algorithm 3 finds its ith vertex by at the latest the jth stage, and has latency at most $\sum_{i=1}^{j} 2\alpha \cdot 2^{l+1} \ln n = 2^{j+3}\alpha \ln n$. This quantity is within a factor of $8\alpha \ln n$ of v's latency, so the overall latency of our solution is at most $8\alpha(\ln n)M_k^*$. □

6. *The Constant-Factor Approximation*

The reason we had to use $k \ln n$ trees earlier was that we were forced to cover at least i vertices. But what if we relax this to say we only need to cover a constant fraction of them? To accomplish this, we will need only $O(k)$ trees, which should be enough. Let's now push through this line of reasoning.

Here is our new algorithm.

Algorithm. 4

For each $j \geq 0$ do:
  For $l = 1, 2, \ldots, k$ do:
    Let $T_l^j = \mathrm{BudgetTree}(2^{j+1})$. Remove the vertices
    of T_l^j other than s from V.
  Arbitrarily give each repairman one of the k trees from this
  stage.
Tell each repairman to traverse Euler tours of his trees in increasing order of j, short-circuiting to avoid going to vertices twice.

For $j \geq 0$, let n_j be the number of vertices in MLT_k^* which have latency more than 2^{j+1}, and let n'_j be the number of vertices yet not covered by the end of step j of Algorithm 4. For convenience, define $n'_{-1} = n$. Observe that these two quantities are what we have already been comparing in the previous three lemmas.

The next lemma says that we have only a small number more uncovered vertices at step j than the optimal solution.

LEMMA 6.1. *For all* $j \geq 0$, $n'_j \leq n_j + \frac{1}{e}(n'_{j-1} - n_j)$.

PROOF. Notice that if $n'_{j-1} \leq n_j$, then $n'_j \leq n'_{j-1} \leq \frac{1}{e}n'_{j-1} + (1 - \frac{1}{e})n_j$, as desired. Hence we may assume in what follows that $n'_{j-1} > n_j$.

Consider stage j of Algorithm 4. Let T_j be MLT_k^* restricted to vertices with latency at most 2^{j+1}. Then T_j is a set of k partial tours. At the beginning of the stage, there are $n - n'_{j-1}$ vertices that have been covered by Algorithm 4. There are $n - n_j$ vertices that have been covered by T_j (both of these statements are by definition).

Consider the situation in stage j where we have chosen l out of the k trees. Let $i_l = (n - n_j) -$ (number of vertices we have covered so far in all stages). Note that $i_0 = (n - n_j) - (n - n'_{j-1}) = n'_{j-1} - n_j$. At that point, there are at least i_l vertices covered by T_j and not yet by our algorithm. These vertices are covered by k trees of cost at most 2^{j+1} each in T_j. Therefore one of them must cover at least i_l/k of them. This means that there is a tree of cost at most 2^{j+1} which covers at least i_l/k more vertices other than the ones which Algorithm 4 has covered so far. Hence $i_{l+1} \leq i_l - i_l/k = i_l(1 - 1/k)$. By transitivity, we can see that $i_l \leq i_0(1 - 1/k)^l$, and in particular that $i_k \leq i_0(1 - 1/k)^k \leq \frac{1}{e}i_0 = \frac{1}{e}(n'_{j-1} - n_j)$.

Hence at the end of stage j we have covered only at most $\frac{1}{e}(n'_{j-1} - n_j)$ fewer total vertices than those in the optimal solution with latency at most 2^{j+1}. We conclude that $n'_j \leq n_j + \frac{1}{e}(n'_{j-1} - n_j)$, as desired. □

LEMMA 6.2. *Assume that the distance from* s *to any vertex is at least* 2. *Then* $\sum_{j\geq 0} 2^j n_j \leq M_k^*$, *and the sum of the latencies of the tours from Algorithm* 4 *is at most* $\sum_{j\geq 0} 2^{j+4}\alpha n'_j$.

PROOF. Let's first prove the first inequality. For each value of $j \geq 0$, construct one imaginary bucket. For each vertex with latency more than 2^{j+1} in the optimal set of tours, contribute a payment of 2^j to bucket j.

Suppose that some vertex v has latency in the range $(2^{j+1}, 2^{j+2}]$ (hence $j \geq 1$, since the distance from s is at least 2). Then it contributes to the buckets 0 through j. The sum of its contributed payments is then $\sum_{i=0}^{j} 2^i = 2^{j+1} - 1$, which is in turn no larger than the latency of v. Hence, each vertex v contributes at most as much to the total amounts in all buckets as its latency. Now notice that the sum, over all vertices, of the values contributed to the buckets is exactly the lefthand side of the inequality. This finishes the proof.

Now for the second inequality. We use a similar imaginary bucket argument here also. Say that a vertex pays $2^{j+4}\alpha$ to bucket j if it is not covered by the jth step of the algorithm. Each vertex v covered in step $j + 1$ has actual latency at most $2^{j+4}\alpha$ (using the same analysis as in the previous sections). Supposing it is covered in that step means that it pays $\sum_{i=0}^{j} 2^{i+4}\alpha = 16(2^{j+1} - 1)\alpha \geq 2^{j+4}\alpha$ (when $j \geq 1$,

which is true, since all vertices are at distance at least $2^1 = 2$). Hence v pays at least as much as its latency. Since the sum over all buckets of the amounts paid is $\sum_{j\geq 0} 2^{j+4}\alpha n'_j$, we are done. □

LEMMA 6.3. *Assuming that all distances between vertices are at least* 2*, Algorithm* 4 *is a* 49.42α*-approximation algorithm for the* k*-traveling repairmen problem.*

PROOF. Let the cost of the tour produced by Algorithm 4 be C. Now $C \leq \sum_j 2^{j+4}\alpha n'_j$ by the previous lemma. Since we want to upper-bound C, we may as well assume that $C = \sum_j 2^{j+4}\alpha n'_j$ since this is only a weaker bound. Also notice that since all intervertex distances are at least 2, $n'_0 = n_0$. For convenience, let $C_\alpha = C/\alpha$. Now, applying Lemma 6.1,

$$\begin{aligned} C_\alpha &= \sum_{j\geq 0} 2^{j+4} n'_j \\ &\leq \sum_{j\geq 0} 2^{j+4}\left(n_j + \frac{1}{\mathrm{e}}(n'_{j-1} - n_j)\right) \qquad (1) \\ &= \sum_{j\geq 0} 2^{j+4}\left(1 - \frac{1}{\mathrm{e}}\right) n_j + \frac{16}{e} n + \sum_{j\geq 0} 2^{j+5}\frac{1}{\mathrm{e}} n'_j \qquad (2) \\ &\leq 16\left(1 - \frac{1}{2e}\right) M_k^* + \frac{2}{\mathrm{e}} C_\alpha. \qquad (3) \end{aligned}$$

Inequalities (1) and (2) are due to Lemma 6.1 and the fact that, by definition, $n'_{-1} = n$, respectively. Inequality (3) is due to Lemma 6.2 plus the fact that since all vertices are at distance at least 2 from s, $M'_k > 2n$. Putting it together, we have that $C_\alpha \leq 16(1 - \frac{1}{2e})M_k^* + \frac{2}{\mathrm{e}}C_\alpha$, and hence that

$$C \leq \frac{16\left(1 - \frac{1}{2e}\right)}{1 - \frac{2}{\mathrm{e}}}\alpha M_k^* = \frac{16e}{\mathrm{e} - 2}\alpha M_k^* \leq 49.42\alpha M_k^*. \qquad \square$$

7. *Improving the Approximation Factor.*

In this section we show how to improve the the approximation factor from 60.56 to 8.497. To do this, we first start with an improved algorithm for the single-repairman problem. Essentially the same analysis has previously been derived by Goemans and Kleinberg [1996]; however, this proof seems simpler and can be directly applied to the Blum et al. algorithm, and to ours as well.

7.1 THE SITUATION WITH ONE REPAIRMAN

LEMMA 7.1. *There exists a* 3.59α*-approximation algorithm for the single-repairman problem.*

PROOF. Let b be a real number greater than 1 (we will fix its exact value later). Let $c = b^U$, where U is a random variable distributed uniformly between 0 and 1. Instead of finding trees of cost at most $1, 2, 4, 8, \ldots$ which cover the most vertices (as do Blum et al.), we will find the trees of cost at most $c, cb, cb^2, \ldots$ which cover the most vertices. Also notice that the Euler tour of each tree can be traversed either

backwards or forwards: We will choose randomly for each tree in which direction to traverse.

Since our algorithm is randomized, the latency of each vertex v' will depend on the value of c. Since the choice of c and of the random Euler tour direction are independent of each other, we can compute the expected latency of v' sequentially by fixing c and taking the expectation over the Euler tour, then finally taking the expectation over c.

Now consider some vertex v which is the ith in the optimal tour. It has latency db^j for some j and d, where $1 \le d < b$. The first tree T that we look for which has objective cost $C \ge db^j\alpha$ will hit at least i vertices. We want to compare the latency of v with the ith vertex v' in our tour. There are two cases, depending on the value of c. If $c < d$, T is the one having objective cost $cb^{j+1}\alpha$. If $c \ge d$, it has objective cost $cb^j\alpha$.

In the first case, the expected latency of our vertex over the random direction of the Euler tour is

$$\frac{1}{2}\left(\left(\sum_{l=0}^{j} 2cb^l\right)\alpha + D\right) + \frac{1}{2}\left(\left(\sum_{l=0}^{j} 2cb^l\right)\alpha + 2cb^{j+1}\alpha - D\right) = \left(\sum_{l=0}^{j} 2cb^l\right)\alpha + cb^{j+1}\alpha,$$

where D is the distance traveled in one direction of the tree to reach v'. The expected distance in the second case can be computed in the same way to be $\sum_{l=0}^{j-1} 2cb^l\alpha + cb^j\alpha$.

Thus, using the fact that $c = b^U$, we compute the expected latency of v' as follows. First notice that $c = b^U < d$, iff $U < \log_b d$. This means that for $U \in [0, \log_b d)$ we are in the first case, and for $U \in [\log_b d, 1]$ we are in the second case. Hence

$$\begin{aligned}
\mathrm{E}_{c,\mathrm{Euler}}[\text{latency of } v'] &= \alpha \int_0^{\log_b d} \left(\left(\sum_{l=0}^{j} 2b^U b^l\right) + b^U b^{j+1}\right) dU \\
&\quad + \alpha \int_{\log_b d}^{1} \left(\left(\sum_{l=0}^{j-1} 2b^U b^l\right) + b^U b^j\right) dU \\
&= \alpha \left(\left(\sum_{l=0}^{j} 2b^l\right) + b^{j+1}\right) \left(\int_0^{\log_b d} b^U dU\right) \\
&\quad + \alpha \left(\left(\sum_{l=0}^{j-1} 2b^l\right) + b^j\right) \left(\int_{\log_b d}^{1} b^U dU\right) \\
&= \alpha \left(\left(\sum_{l=0}^{j} 2b^l\right) + b^{j+1}\right) \left(\frac{d-1}{\ln b}\right) \\
&\quad + \alpha \left(\left(\sum_{l=0}^{j-1} 2b^l\right) + b^j\right) \left(\frac{b-d}{\ln b}\right) \\
&= \alpha \left(\frac{b^{j+1} + b^j - 2}{\ln b}\right) + \alpha \left(\frac{(d-1)(b^{j+1} + b^j)}{\ln b}\right).
\end{aligned}$$

Therefore, the expected ratio between the latency of the ith vertex in the optimal tour and the latency in our tour is

$$\begin{aligned} \mathrm{E}_{c,\mathrm{Euler}}[\mathrm{ratio}] &= \alpha \frac{\left(\frac{b^{j+1}+b^j-2}{\ln b}\right) + \left(\frac{(d-1)(b^{j+1}+b^j)}{\ln b}\right)}{db^j} \\ &\le \alpha \left(\frac{1}{d}\right)\left(\frac{b+1}{\ln b}\right) + \left(\frac{d-1}{d}\right)\left(\frac{b+1}{\ln b}\right) \\ &= \alpha \frac{b+1}{\ln b}. \end{aligned} \tag{4}$$

Now let's optimize for the best b. Setting the derivative of this quantity equal to zero, we get $b = \frac{b+1}{\ln b}$, which has $b = 3.59\ldots$ as its solution. (In this situation, the approximation ratio and the base of exponentiation are the same.) □

7.1. THE SITUATION WITH k REPAIRMEN. In this subsection, we extend this idea to the case with k repairmen.

Fix $b > 1$ to be determined later. Here is our new algorithm.

Algorithm. 5

Choose $c = b^U$ according to the random variable U,
which has a uniform distribution on [0, 1]. For each $j \ge 0$:
  For $l = 1, \ldots, k$
    Let T_l^j = BudgetTree(cb^j). Remove the vertices
    of T_l^j other than s from the graph.
  Arbitrarily give each repairman one of the k trees from this
  stage.

Tell each repairman to traverse his trees in increasing order of j, and to traverse each tree either in the forward Euler tour direction or the backward Euler tour direction according to the flip of an unbiased coin.

In order to figure out how well this works, we proceed in almost the same manner as in the previous case. However, instead of comparing the cost C of our solution directly with the optimal value M_k^*, we introduce another random variable $M_k'^*$, which is the lower bound on the optimal cost under our "charging scheme," and analyze the ratio between the expected values of C and $M_k'^*$. Then we bound the expected ratio between M_k^* and $M_k'^*$. Putting these two bounds together yields an upper bound on the expected approximation factor.

Let q_j be the number of vertices in MLT_k^* which have latency more than cb^j. Let q'_j be the number of vertices not yet covered by the end of step j of Algorithm 5. For convenience, define $q'_{-1} = 0$. We define $M_k'^*$ to be $\sum_{j\ge 0} c(b-1)b^{j-1}q_j$. We note that q_j and q'_j are random variables over the choice of c.

Now we develop new versions of Lemmas 6.1 and 6.2.

LEMMA 7.2 [REPLACEMENT FOR LEMMA 6.1]. *For all* $j \ge j_0$, $q'_j \le q_j + \frac{1}{\mathrm{e}}(q'_{j-1} - q_j)$.

PROOF. For each choice of c, the lemma follows from the same arguments as in Lemma 6.1. Hence it follows for the random variables also. □

LEMMA 7.3 [REPLACEMENT FOR LEMMA 6.2]. *Fix a choice of* c. *Let* $b > 1$ *and* $c'' > 1$ *be real numbers, and let* $c' = c''(b-1)c$. *Assume that all vertices are at least a*

distance of b^{j_0+1} *away from* s*, where* $j_0 \geq \lceil \log_b \frac{c''}{c''-1} \rceil$*. Then* $\sum_{j\geq 0}(b-1)cb^{j-1}q_j = M_k'^* \leq M_k^*$*. Also, the expected (over Euler tour directions) sum of the latencies of tours from Algorithm* 5 *is at most* $\sum_{j\geq 0}(2c+c')b^j\alpha q'_j$*.*

PROOF. The proof of the first inequality is straightforward. Say that a vertex pays $(b-1)cb^{j-1}$ into bucket j if it has latency more than cb^j. Consider some vertex v with latency db^j for some $1 \leq d < b$ and $j \geq j_0$. Then if $c < d$, it pays into the buckets 0 through j. In this case, its total contributed payments are

$$\sum_{i=0}^{j}(b-1)cb^{i-1} = \frac{(b-1)c}{b}\frac{b^{j+1}-1}{b-1} = \frac{c}{b}(b^{j+1}-1) = cb^j - \frac{c}{b} < db^j.$$

If $c \geq d$, it pays into the buckets 0 through $j-1$. In this case, its total contributed payments are

$$\sum_{i=0}^{j-1}(b-1)cb^{i-1} = \frac{(b-1)c}{b}\frac{b^{j}-1}{b-1} = \frac{c}{b}(b^{j}-1) = cb^{j-1} - \frac{c}{b} \leq db^j.$$

To finish the proof, note that the sum over all vertices of the values contributed to the buckets is the lefthand side of the inequality.

To prove the second inequality, now say that a vertex pays $(2c+c')b^j\alpha$ into bucket j if it is not covered by step j. Each vertex v covered in step $j+1$ has expected (over the Euler tour direction in the last tour) actual latency at most

$$\begin{aligned}&\frac{1}{2}\left(\left(\sum_{i=0}^{j}2cb^i\right)\alpha + D\right) + \frac{1}{2}\left(\left(\sum_{i=0}^{j}2cb^i\right)\alpha + (2cb^{j+1}\alpha - D)\right)\\ &\quad = \left(\sum_{i=0}^{j}2cb^i\right)\alpha + cb^{j+1}\alpha, \qquad (5)\end{aligned}$$

where D is its distance along the Euler tour of the step $j+1$ tree that contains v.

On the other hand, a vertex v covered in step $j+1$ will have to pay $\sum_{i=0}^{j}(2c+c')\alpha b^i = \sum_{i=0}^{j}2cb^i\alpha + \sum_{i=0}^{j}c'b^i\alpha$ to the buckets 0 through j. Since $\sum_{i=0}^{j}2cb^i\alpha$ is exactly the summation in the righthand side of Eq. (5), we only need to show that $\sum_{i=0}^{j}c'b^i\alpha$ is at least as large as the second term following the summation. Indeed

$$\sum_{i=0}^{j}c'b^i\alpha = c'\sum_{i=0}^{j}b^i\alpha = c'\frac{b^{j+1}-1}{b-1}\alpha = c''cb^{j+1}\alpha - c''c\alpha,$$

which we want to be larger than than $cb^{j+1}\alpha$. Hence we need

$$j \geq \log_b \frac{c''}{c''-1} - 1.$$

We can achieve this by setting $j_0 \geq \lceil \log_b \frac{c''}{c''-1} \rceil$, and the proof follows. □

Next we need the lemma that relates the optimal solution and our lower bound. It comprises basically the same analysis as that in Lemma 7.1.

LEMMA 7.4. *If all vertices are at least a distance of* b^{j_0+1} *away from* s*, where* $j_0 \geq 1$*, then* $E[M_k'^*] \leq \frac{b-1}{b\ln b}M_k^*$*.*

PROOF. Consider any vertex v having a latency of db^j in the optimal tour, where $1 \le d < b$. From Lemma 7.3, observe that if $c < d$, we charge $\sum_{i=0}^{j}(b-1)cb^{i-1}$ to $M_k'^*$; otherwise, we charge $\sum_{i=0}^{j-1}(b-1)cb^{i-1}$. Thus the expected contribution of v is

$$\begin{aligned}
&(b-1)\left[\int_0^{\log_b d}\left(\sum_{i=0}^{j} b^U b^{i-1}\right)dU + \int_{\log_b d}^{1}\left(\sum_{i=0}^{j-1} b^U b^{i-1}\right)dU\right] \\
&\quad = (b-1)\left[\left(\frac{b^{j+1}-1}{b(b-1)}\right)\left(\frac{d-1}{\ln b}\right) + \left(\frac{b^j-1}{b(b-1)}\right)\left(\frac{b-d}{\ln b}\right)\right] \\
&\quad = \frac{(b-1)(b^j d-1)}{b\ln b}
\end{aligned}$$

Since $\mathrm{E}[M_k'^*] = \sum_v \mathrm{E}[\text{contribution of } v]$, and letting $d(v)$ and $j(v)$ be such that the latency of vertex v in the optimal solution is $d(v)b^{j(v)}$ so that $M_k^* = \sum_v d(v)b^{j(v)}$, we know that $\mathrm{E}[M_k'^*]/M_k^* \le \max_v \mathrm{E}[\text{contribution of } v]/(d(v)b^{j(v)})$. But for any v, this last quantity is

$$\frac{(b-1)\frac{b^j d-1}{b\ln b}}{db^j} \le \frac{b-1}{b\ln b},$$

which proves the lemma. □

Now we can restate the main lemma.

LEMMA 7.5 [REPLACEMENT FOR LEMMA 6.3]. *Fix $b = 1.616$, let c' and c'' be as in Lemma 7.3, and all vertices be a distance at least b^{j_0+1} away from s, where $j_0 = \lceil \log_b \frac{c''}{c''-1} \rceil > 0$. Then Algorithm 5 gives an 8.497α-approximation for the k-traveling repairmen problem.*

PROOF. Fix c for now.

Let the expected cost (over Euler tour directions) of the tour produced by Algorithm 5 be C. Let C'_α, an upper bound of our cost divided by α, be $\sum_{j\ge 0}(2c+c')b^j q'_j$, where $c' = c''(b-1)c$. Since by Lemma 7.3 we know that $C \le C'_\alpha \alpha$, we will use an upper bound on C'_α to get an upper bound on C. Then

$$\begin{aligned}
C'_\alpha &= \sum_{j\ge 0}(2c+c')b^j q'_j \\
&\le \sum_{j\ge 0}(2c+c')b^j\left(q_j + \frac{1}{\mathrm{e}}(q'_{j-1} - q_j)\right) \qquad (6)\\
&= \left(1-\frac{1}{\mathrm{e}}\right)\sum_{j\ge 0}(2c+c')b^j q_j + \sum_{j\ge 1}(2c+c')b^j\frac{1}{\mathrm{e}}q'_{j-1} \\
&= \left(1-\frac{1}{\mathrm{e}}\right)b(2+c''(b-1))\sum_{j\ge 0} cb^{j-1}q_j + \frac{b}{\mathrm{e}}\sum_{j\ge 0}(2c+c')b^j q'_j \\
&\le \left(1-\frac{1}{e}\right)\frac{b(2+c''(b-1))}{b-1}M_k'^* + \frac{b}{\mathrm{e}}C'_\alpha, \qquad (7)
\end{aligned}$$

where transformations (6) and (7) are due to Lemmas 7.2 and 7.3 (this is where the $b-1$ in the denominator appears), respectively. This implies that

$$C \leq \left(\frac{b(2+c''(b-1))\cdot\left(1-\frac{1}{\mathrm{e}}\right)}{(b-1)(1-b/\mathrm{e})}\right)\alpha M_k'^*$$

if $b < \mathrm{e}$. Hence

$$\mathrm{E}_c[C] \leq \left(\frac{b(2+c''(b-1))\cdot\left(1-\frac{1}{\mathrm{e}}\right)}{(b-1)(1-b/\mathrm{e})}\right)\alpha \mathrm{E}_c[M_k'^*];$$

using Lemma 7.4,

$$\begin{aligned}\mathrm{E}[C] &\leq \left(\frac{b(2+c''(b-1))\cdot\left(1-\frac{1}{\mathrm{e}}\right)}{(b-1)(1-b/\mathrm{e})}\right)\cdot\left(\frac{b-1}{b\ln b}\right)\alpha M_k^* \\ &= \left(\frac{(2+c''(b-1))}{\ln b}\frac{\mathrm{e}-1}{e-b})\right)\alpha M_k^*. \end{aligned} \tag{8}$$

Note that by increasing j_0, we can make the value of c'' in the aforesaid expression arbitrarily close to 1. It is interesting to observe that if we set $c''=1$ term, then Eq. (8) becomes

$$\frac{b+1}{\ln b}\frac{\mathrm{e}-1}{\mathrm{e}-b}\alpha, \tag{9}$$

which can be compared with (4).

Setting $c''=1+\epsilon$, we can set $j_0 = \lceil \log_b \frac{c''}{c''-1}\rceil = \lceil \log_b \frac{1+\epsilon}{\epsilon}\rceil = O(\log_b \frac{1}{\epsilon})$ to get an approximation factor arbitrarily close to Eq. (9). Optimizing over the value of b gives us that $b \approx 1.616$ is the optimal value, which results in a final approximation factor of $(8.4965\ldots+\epsilon')$, where ϵ' can be made as small as desired. Hence we can easily get an 8.497-approximation, which concludes the lemma. □

Yet another optimization could be to choose Nk trees instead of k in each step of Algorithm 5, and randomly assign each repairman a $1/k$ fraction of the trees. If $N < 1$, then this makes the expected cost of each repairman's trees less, but increases the $1/e$ factor in Lemma 7.2. If $N > 1$, then the first quantity is increased but the second is decreased. By computing a derivative of the induced cost function as a function of N, we can find the optimal value.

This optimization helped the approximation factor in earlier versions of the article which had a less sophisticated analysis. In the present version, however, $N = 1$ appears to be the optimal value. We point this out since future improvements may make this a valuable optimization once again.

ACKNOWLEDGMENTS. We would like to thank Kunal Talwar for insightful discussions and a pointer to previous work on randomized scaling in Chekuri et al. [2002], Sanjeev Arora for pointing us toward Haimovich et al. [1988], and the anonymous referees for helpful comments and clarifying suggestions for both the conference and journal version of this article.

REFERENCES

ARCHER, A., LEVIN, A., AND WILLIAMSON, D. P. 2003. A faster, better approximation algorithm for the minimum latency problem. Tech. Rep. 1362, Cornell University.

ARCHER, A., AND WILLIAMSON, D. 2003. Faster approximation algorithms for the minimum latency problem. In *Proceedings of the 14th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*.

ARORA, S., AND KARAKOSTAS, G. 1999. Approximation schemes for minimum latency problems. In *Proceedings of the 31st Annual ACM Symposium on Theory of Computing (STOC)*.

ARORA, S., AND KARAKOSTAS, G. 2000. A 2+epsilon approximation for the k-MST problem. In *Proceedings of the 11th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*.

ASANO, T., KATOH, N., TAMAKI, H., AND TOKUYAMA, T. 1997. Covering points in the plane by k-tours: Towards a polynomial time approximation scheme for general k. In *Proceedings of the 29th Annual ACM Symposium on Theory of Computing (STOC)*, 275–283.

BLUM, A., CHALASANI, P., COPPERSMITH, D., PULLEYBLANK, B., RAGHAVAN, P., AND SUDAN, M. 1994. The minimum latency problem. In *Proceedings of the 26th Annual ACM Symposium on Theory of Computing (STOC)*, 163–171.

CHARIKAR, M., KHULLER, S., AND RAGHAVACHARI, B. 2001. Algorithms for capacitated vehicle routing. *SIAM J. Comput. 31,* 3, 665–682. (Conference version in STOC 1998).

CHAUDRY, K., GODFREY, B., RAO, S., AND TALWAR, K. 2003. Paths, trees, and minimum latency tours. In *Proceedings of the 44th Annual Symposium on Foundations of Computer Science (FOCS)*, 36–45.

CHEKURI, C., GUPTA, A., KUMAR, A., NAOR, S., AND RAZ, D. 2002. Building edge-failure resilient networks. In *Proceedings of the 9th Conference on Integer Programming and Combinatorial Optimization (IPCO)*, 439–456.

CHEKURI, C., AND KUMAR, A. 2004. Maximum coverage problem with group budget constraints and applications. `http://cm.bell-labs.com/cm/cs/who/chekuri/postscript/coverage.ps`.

CHRISTOfiDES, N. 1985. *The Traveling Salesman Problem*. John Wiley, New York, 431–448.

CORMEN, T. H., LEISERSON, C. E., AND RIVEST, R. L. 1990. *Introduction to Algorithms*. MIT Press/McGraw-Hill.

GARG, N. 1996. A 3-approximation for the minimum tree spanning k vertices. In *Proceedings of the 37th Annual Symposium on Foundations of Computer Science (FOCS)*, 302–309.

GOEMANS, M., AND KLEINBERG, J. 1996. An improved approximation ratio for the minimum latency problem. In *Proceedings of the 7th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*.

HAIMOVICH, M., KAN, A. R., AND STOUGIE, L. 1988. Analysis for heuristics for vehicle routing problems. In *Vehicle Routing: Methods and Studies*. Elsevier Science, Chapter 3.

SAHNI, S., AND GONZALES, T. 1976. P-complete approximation problems. *J. ACM 23,* 3, 555–565.

SITTERS, R. 2002. The minimum latency problem is NP-hard for weighted trees. In *Proceedings of the 9th Conference on Integer Programming and Combinatorial Optimization*, 230–239.

TOTH, P., AND VIGO, D., eds. 2002. *The Vehicle Routing Problem*. SIAM Monographs on Discrete Mathematics and Applications. Society for Industrial and Applied Mathematics, Philadelphia, PA.

RECEIVED JANUARY 2004; ACCEPTED JUNE 2006