SIAM J. COMPUT.
Vol. 37, No. 5, pp. 1472–1498
© 2008 Society for Industrial and Applied Mathematics

A FASTER, BETTER APPROXIMATION ALGORITHM FOR THE MINIMUM LATENCY PROBLEM*

AARON ARCHER†, ASAF LEVIN‡, AND DAVID P. WILLIAMSON§

Abstract. We give a 7.18-approximation algorithm for the minimum latency problem that uses only $O(n \log n)$ calls to the prize-collecting Steiner tree (PCST) subroutine of Goemans and Williamson. This improves the previous best algorithms in both performance guarantee and running time. A previous algorithm of Goemans and Kleinberg for the minimum latency problem requires an approximation algorithm for the k-minimum spanning tree (k-MST) problem which is called as a black box for each value of k. Their algorithm can achieve an approximation factor of 10.77 while making $O(n(n + \log C) \log n)$ PCST calls, a factor of 8.98 using $O(n^3(n + \log C) \log n)$ PCST calls, or a factor of $7.18 + \epsilon$ using $n^{O(1/\epsilon)} \log C$ PCST calls, via the k-MST algorithms of Garg, Arya and Ramesh, and Arora and Karakostas, respectively. Here n denotes the number of nodes in the instance, and C is the largest edge cost in the input. In all cases, the running time is dominated by the PCST calls. Since the PCST subroutine can be implemented to run in $O(n^2)$ time, the overall running time of our algorithm is $O(n^3 \log n)$. We also give a faster randomized version of our algorithm that achieves the same approximation guarantee in expectation, but uses only $O(\log^2 n)$ PCST calls, and derandomize it to obtain a deterministic algorithm with factor $7.18 + \epsilon$, using $O(\frac{1}{\epsilon} \log^2 n)$ PCST calls. The basic idea for our improvement is that we do not treat the k-MST algorithm as a black box. This allows us to take advantage of some special situations in which the PCST subroutine delivers a 2-approximate k-MST. We are able to obtain the same approximation ratio that would be given by Goemans and Kleinberg if we had access to 2-approximate k-MSTs for all values of k, even though we have them only for some values of k that we are not able to specify in advance. We also extend our algorithm to a weighted version of the minimum latency problem.

Key words. minimum latency problem, traveling repairman, approximation algorithm, Lagrangian relaxation, prize-collecting Steiner tree

AMS subject classifications. 68W25, 90C27, 90C59

DOI. 10.1137/07068151X

1. Introduction.

Given a metric space with n nodes and a tour starting at some node and visiting all of the others, the latency of node v is defined to be the total distance traveled before reaching v. The *minimum latency problem* (MLP) asks for a tour, starting at a specified root r and visiting all nodes, such that the total latency is minimized. This problem is sometimes called the *traveling repairman problem* or the *deliveryman problem* and has been well studied in both the computer science and the operations research literature. The MLP models routing problems in which one wants to minimize the average time each customer (node) has to wait before being served (visited) rather than the total time to visit all nodes, as in the case of the famous

*Received by the editors January 31, 2007; accepted for publication (in revised form) August 29, 2007; published electronically January 30, 2008. An extended abstract of this paper previously appeared in *Proceedings of the* 14*th Annual ACM-SIAM Symposium on Discrete Algorithms*, 2003, pp. 88–96. An improved version appeared as Cornell ORIE Technical report 1362, 2003.
http://www.siam.org/journals/sicomp/37-5/68151.html

†Algorithms and Optimization Department, AT&T Shannon Research Laboratory, 180 Park Avenue, Florham Park, NJ 07932 (aarcher@research.att.com). Much of this research was performed while visiting the IBM Almaden Research Center, and part was supported by the Fannie and John Hertz Foundation while the author was at Cornell University.

‡Department of Statistics, The Hebrew University, Jerusalem 91905, Israel (levinas@mscc.huji.ac.il).

§School of Operations Research and Industrial Engineering, Cornell University, Ithaca, NY 14853 (dpw@cs.cornell.edu). Most of this research was performed while this author was a Research Staff Member at the IBM Almaden Research Center.

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

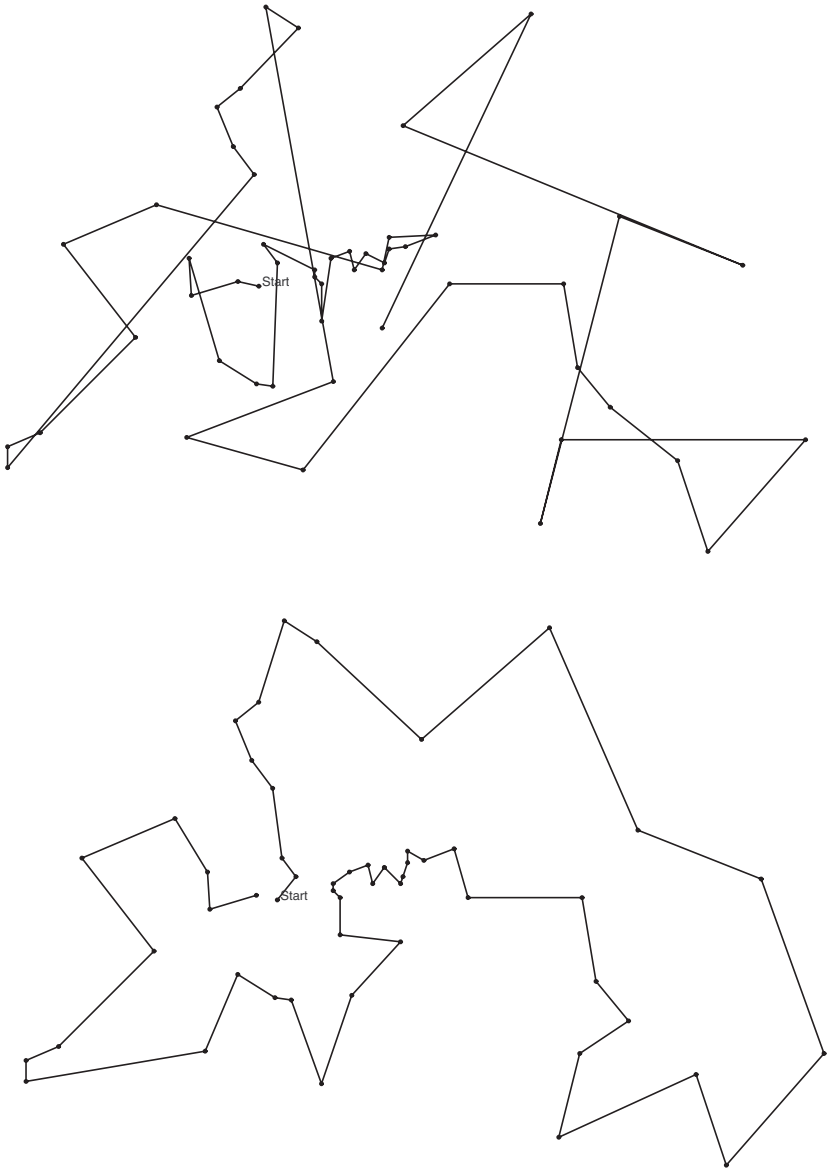

FIG. 1. *Comparison of our algorithm (top) versus the optimal TSP tour (bottom) on the berlin52 instance.*

traveling salesman problem (TSP). In this sense, the MLP takes a customer-oriented view, whereas the TSP is server-oriented.

As one illustration of how different the MLP is from the TSP, in Figure 1 we show a comparison of tours for a 2-dimensional Euclidean instance. This is the berlin52 instance, taken from the TSPLIB [33]. At the top we show the MLP tour produced by the algorithm we develop in this paper, and below that we show the optimal TSP tour for this instance. Note that the tour produced by the MLP algorithm attempts to visit most of the nodes near the start node first, before making its way to the more distant nodes. On this instance, the total latency of the optimal TSP tour is 4.48 times as large as that of the tour given by our algorithm.

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

Koutsoupias, Papadimitriou, and Yannakakis [25] and Ausiello, Leonardi, and Marchetti-Spaccamela [8] motivate the MLP in terms of searching a graph (such as the Web) to find a hidden treasure. If the treasure is equally likely to reside at any node of the graph, then the optimal MLP tour minimizes the expected time to find it. If the location of the treasure is given by a nonuniform probability distribution, the problem reduces to the weighted minimum latency problem that we define below.

The MLP was shown to be NP-hard for general metric spaces by Sahni and Gonzalez [34]. It is also Max-SNP hard, by a reduction from TSP(1,2) (the traveling salesman problem with all distances restricted to be either 1 or 2) [31, 11]. Therefore, there is no polynomial-time approximation scheme for the MLP on general metric spaces unless P = NP. Sitters showed that the problem is NP-hard even for weighted trees [36]. On the positive side, Arora and Karakostas gave quasi-polynomial-time approximation schemes for the MLP on weighted trees and constant dimensional Euclidean spaces [5]. Blum et al. gave the first constant factor approximation algorithm for general metric spaces [11], which was later improved by Goemans and Kleinberg [22]. We elaborate on these results below.

Much work has focused on exact (exponential time) solution approaches to the MLP [35, 16, 10, 28, 42] and on the more general time-dependent traveling salesman problem (TDTSP) [39, 32]. In the TDTSP, the distance between the ith and $(i+1)$st nodes in the traveling salesman tour is multiplied by some weight $w(i)$ in the objective function. The ordinary TSP is the case where all $w(i) = 1$; the MLP is the case where $w(i) = n - i$. The time-dependent orienteering problem (considered in [17]) is dual to the TDTSP—the salesman aims to maximize the number of nodes visited before a given deadline, given that travel times vary as in the TDTSP. Various heuristics for the MLP are evaluated in [38, 40], while [2] analyzes a stochastic version of the problem. In the online variant [15, 26], new nodes appear in the graph as the repairman is traveling. Many authors have considered special cases of the MLP, where the metric is given by an underlying network with some special structure [1, 30, 9, 37, 41]. Fakcharoenphol, Harrelson, and Rao gave a constant factor approximation algorithm for the variant where there are k repairmen who must collectively visit all of the nodes [14].

Because the MLP is NP-hard, we shall consider approximation algorithms for the problem. An *α-approximation algorithm* runs in polynomial time and produces a solution with total latency no more than α times the total latency of a minimum latency tour. The value of α is called the *performance guarantee* of the algorithm.

The first constant factor approximation algorithm for the problem was given by Blum et al. [11]. They also show how to use a β-approximation algorithm for the rooted k-minimum spanning tree (k-MST) problem as a black box and convert it into an 8β-approximation algorithm for the MLP. In the k-MST problem, we are given a graph with costs on the edges and must find the minimum-cost tree spanning at least k nodes. In the *rooted* version, the tree must contain some specified root r.[1] The connection between the k-MST problem and the MLP is that the cost of the optimal k-MST rooted at r is a lower bound on the latency of the kth node visited by the optimal MLP tour. Goemans and Kleinberg (GK) [22] subsequently improved the

[1] Although there has been some confusion in the literature on this point, the rooted and unrooted versions of the k-MST problem can each be reduced to the other as follows. To solve the unrooted version, one can run an algorithm for the rooted version with each of the n possible choices for the root and take the cheapest solution. To solve the rooted version, one can generate an unrooted instance where there are an extra n nodes, each connected to the root by an edge of zero cost, and ask for an $(n+k)$-MST. The reduction works because every tree spanning $n+k$ nodes in the modified instance includes the root.

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

performance guarantee of the algorithm of Blum et al. to 3.59β. When preliminary versions of our work appeared [4, 3], the best approximation algorithms known for the k-MST problem were a 3-approximation by Garg [19], a 2.5-approximation by Arya and Ramesh [7], and a $(2+\epsilon)$-approximation by Arora and Karakostas [6], yielding MLP guarantees of 10.77, 8.98, and $7.18+\epsilon$, respectively. Each improvement in the approximation guarantee came at the cost of a significant increase in the running time of the algorithm.

In this paper we further explore the connection between the MLP and rooted k-MST problems. We obtain a performance guarantee of 7.18, slightly improving the previous best of $7.18+\epsilon$. Moreover, our algorithm also has a running time that is faster than the GK algorithm using any of the k-MST algorithms above. In each of these algorithms, the running time is dominated by multiple subroutine calls to an algorithm of Goemans and Williamson for the prize-collecting Steiner tree (PCST) problem [21]. The GK algorithm using Garg's 3-approximation as a subroutine requires $O(n(n+\log C)\log n)$ PCST calls, where C is the cost of the most expensive edge. Using Arya and Ramesh it requires $O(mn(n+\log C)\log n)$, if the metric space is the shortest path metric in a graph with m edges. For general metric spaces, the m becomes an n^2. Using Arora and Karakostas it requires $n^{O(\frac{1}{\epsilon})}\log C$ calls. For each of these algorithms, the $\log C$ factor comes from binary searching for a particular parameter, and by using Megiddo's parametric search technique [29] we can obtain a strongly polynomial version where the $\log C$ is replaced by an n^2. Our algorithm requires only $O(n\log n)$ PCST calls, and we also devise a randomized algorithm that achieves the same approximation factor (in expectation) and uses only $O(\log^2 n)$ PCST calls. This randomized algorithm can be derandomized to obtain a deterministic algorithm with factor $7.18+\epsilon$ using only $O(\frac{1}{\epsilon}\log^2 n)$ PCST calls.

Goemans and Williamson showed how to implement their PCST algorithm in $O(n^2\log n)$ time. Later work of Gabow and Pettie [18] improves this to $O(n^2)$. Thus, our deterministic algorithm runs in time $O(n^3\log n)$ overall, our randomized algorithm in time $O(n^2\log^2 n)$, and our deterministic $(7.18+\epsilon)$-approximation in time $O(\frac{1}{\epsilon}n^2\log^2 n)$.

The main idea in achieving our result is that we do not treat the k-MST algorithm as a black box. It can be shown that the PCST algorithm returns k-MSTs of cost no more than twice optimal for some values of k that depend on the instance and cannot be specified by the algorithm [13]. We refer to a tree spanning k nodes with cost no more than α times the optimal k-MST as an *α-approximate* k-MST. If we had 2-approximate k-MSTs for all $k=2,\ldots,n$, then we could run the GK algorithm, which uses the costs of the trees to select some subset of them to concatenate into an MLP tour. Our trick is to successfully bluff the GK algorithm. We *pretend* to have trees of all sizes by interpolating the costs of the trees we do have to fill in the tree costs for the missing values of k. We refer to these as "phantom" trees. We then prove that, if the GK algorithm were to be run with both real and phantom trees, it would never choose any of the phantom trees to concatenate, so it never calls our bluff. Since the GK algorithm would never select any of the phantom trees anyway, it suffices to use only the real trees we generated. For the analysis to go through, we must also carefully extend our k-MST lower bounds to the phantom values of k. To do this, we utilize the fact that the PCST problem is a Lagrangian relaxation of the k-MST problem, as observed by Chudak, Roughgarden, and Williamson in [13].

The improvement in our running time derives from two sources. The first is that the k-MST algorithms expend a significant amount of work in producing a near-optimal solution of size *exactly* k. In contrast, the workhorse of our algorithms is a

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

TABLE 1

A summary of the running times and approximation ratios of our deterministic and randomized algorithms for the weighted and unweighted MLP. Here $\gamma \approx 3.5912$ is the unique solution to $\gamma \ln \gamma = \gamma + 1$.

MLP version	Approx. ratio	Randomized or deterministic	# PCST calls
Unweighted	2γ	det.	$O(n \log n)$
		rand.	$O(\log^2 n)$
	$2\gamma(1+\epsilon)$	det.	$O(\frac{1}{\epsilon} \log^2 n)$
Weighted	2γ	det.	$O(n \log^2 W)$ or $O(n^3 \log^2 n)$
		rand.	$O(\log^2 W)$ or $O(n^2 \log^2 n)$
	$2\gamma(1+\epsilon)$	det.	$O(\frac{1}{\epsilon} \log^2 W)$ or $O(\frac{1}{\epsilon} n^2 \log^2 n)$

subroutine that generates a pair of near-optimal trees of sizes k_{lo} and k_{hi} sandwiching a given target value k, along with convenient lower bounds on the cost of the optimal k-MST for every $k \in [k_{\text{lo}}, k_{\text{hi}}]$. This sandwiching operation can be done much more rapidly than the k-MST algorithms. Second, the GK algorithm needs to run the k-MST black box $(n-1)$ times, once for each value of k. The Lagrangian relaxation technique we use to generate our k-MST lower bounds allows us to reduce this to $O(\log n)$ sandwiching operations in our randomized algorithm. Our work supplies a nice example where an improved analysis technique has led directly to faster and better algorithms. See Table 1.

While Lagrangian relaxation has a rich history as an effective computational technique (for example, in large scale linear programming), only relatively recently has it also been used as a technique to design and analyze approximation algorithms, starting with [24, 13]. Our work represents another early contribution to this emerging body of research.

Our algorithms also extend to the *weighted* minimum latency problem, previously considered in [5]. In this variant, each node v has an associated positive integer weight w_v, and the goal is to find a tour that minimizes $\sum_v w_v \ell_v$, where ℓ_v is the latency of the node v in the tour. Note that this is equivalent to replacing each node v with a clique of w_v nodes joined by edges of cost zero. Let $W = \sum_v w_v$ (which is always at least n, since each weight is a positive integer). The PCST routine may be trivially modified to run in $O(n^2)$ time (that is, time which depends only on the number of nodes in the original graph and not on W). However, using this reduction, our deterministic algorithm requires $O(W \log W)$ PCST calls, which is only a pseudopolynomial running time. Our randomized algorithm applied to the weighted case makes only $O(\log^2 W)$ PCST calls and hence is weakly polynomial. We show how to alter our randomized algorithm to use $O(n^2 \log^2 n)$ PCST calls, making it strongly polynomial. These algorithms for the weighted MLP have an approximation guarantee of 7.18 in expectation, just like our algorithm for the unweighted version. Each of the algorithms can be derandomized to achieve a deterministic algorithm with the same guarantee, at the expense of blowing up the running time by a factor of n. Alternatively, we can derive a deterministic algorithm with a factor of $7.18 + \epsilon$ while blowing up the running time by a factor of only $\frac{1}{\epsilon}$.

Since the appearance of an extended abstract of this paper [4] and a subsequent improved version [3], Chaudhuri, Godfrey, Rao, and Talwar [12] have given a 3.59-approximation algorithm for the MLP. Instead of k-MSTs, they consider *k-strolls*, which are paths starting at the root and visiting k nodes. Clearly, the length of the shortest k-stroll also gives a lower bound on the latency of the kth node in any tour. They give an algorithm that for some values of k finds a tree spanning k nodes whose

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

cost is no worse than that of the best k-stroll. By invoking our framework above, they are able to reduce the performance guarantee by the factor of 2 that our algorithm incurred by using 2-approximate k-MSTs. Their algorithm uses a slightly modified version of the PCST algorithm as its subroutine but requires guessing the last node in the k-stroll and trying all guesses. Thus, their algorithm requires an extra factor of n in the running time for each tree they generate.

Also after the initial publication of our results, Garg published a 2-approximation algorithm for the k-MST problem [20]. While Garg does not state a running time, the straightforward analysis of his algorithm as he presents it would lead to a running time of $O(\frac{1}{k}mn^2)$ PCST calls. However, we can show that a somewhat different implementation requires only $O(\frac{1}{k}n(n + \log C))$ PCST calls. Using this algorithm as a subroutine for the GK algorithm ties our approximation guarantee but requires $O(n(n + \log C) \log n)$ PCST calls, so our algorithm is still superior.

One might hope that the techniques from [20] and [12] could be combined to obtain a tree spanning k nodes whose cost is no greater than that of the optimal k-stroll, for any value of k chosen by the algorithm rather than just some "lucky" values of k as in [12]. However, as Garg explains in [20], each of these techniques relies on a certain piece of slack in the analysis of the PCST algorithm and each uses up the entire slack, so they cannot be combined. Therefore, even in light of Garg's new result, our techniques are still necessary to obtain the 3.59-approximation algorithm in [12] for the MLP, which is the best approximation ratio achieved to date. Without our techniques, the best approximation ratio that is known would be $3.59 + \epsilon$ from [12], which requires $n^{O(\frac{1}{\epsilon})} \log C$ PCST calls.

The techniques introduced by this paper have also been successfully applied by Hassin and Levin to the minimum latency set cover problem [23] and by Lin, Nagarajan, Rajaraman, and Williamson [27] to develop a framework for designing incremental approximation algorithms. Thus, our techniques have broader applicability than just to the MLP.

The paper is structured as follows. In section 2, we review the main ideas of previous approximation algorithms for the MLP. Section 3 gives our deterministic algorithm, assuming we can generate approximate k-MSTs satisfying certain conditions, and section 4 analyzes the performance guarantee of this algorithm. Section 5 motivates the conditions of section 3 and shows how to satisfy them using something we call our *critical search* primitive, which relies on Lagrangian relaxation. We include a discussion of how one could implement the critical search primitive using Megiddo's parametric search technique [29]. Section 6 shows how to exploit some slack in the previous analysis in order to accelerate the running time, given a slightly different critical search primitive that uses binary search instead of Megiddo's parametric search, although the details of this faster critical search primitive are deferred to section 9. In section 7, we give our faster randomized algorithm, which requires a somewhat different analysis than the deterministic algorithm. We also show how to derandomize it. Section 8 discusses the weighted version of the problem and gives extensions of our deterministic and randomized algorithms to the weighted case. In section 10, we show some experimental results of our algorithm for the unweighted case of MLP. We conclude in section 11 with some thoughts about approaches for further improvements.

2. Intuition and overview. We now describe the basic ideas behind the Blum et al. [11] and GK [22] algorithms and how our approach departs from them. Both analyses use the cost of the optimal k-MST as a lower bound for the latency of the kth

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

node visited in the optimal MLP tour, and both algorithms start with β-approximate solutions to the k-MST problem rooted at r for $k = 2, 3, \ldots, n$. They then select a subsequence of these trees with geometrically increasing costs and concatenate them to get a solution for the MLP. For the sake of intuition, let us assume throughout this section that the sets of nodes spanned by these trees are nested, which turns out to be the worst case for the analysis.

Without loss of generality, the cost of the k-MSTs increases with k. The Blum et al. algorithm buckets the trees according to their cost—for each integer ℓ, it selects the most expensive tree with cost in $(2^\ell, 2^{\ell+1}]$. It doubles each of the selected trees, shortcuts it to make a cycle rooted at r, and then traverses all of these cycles in order, shortcutting nodes it has already visited. Since the last tree selected spans all of the nodes, so does the resulting MLP tour. They compare the latency of the kth node visited in the tour to the cost of the optimal k-MST. They upper bound the latency of the kth node by the total cost of all of the concatenated cycles up to and including the first one that visits this node. They lose a factor of β because the trees are β-approximate k-MSTs, a factor of 2 from the bucketing ratio, a factor of 2 from doubling the trees to get cycles, and a factor of 2 from the geometric sum. This yields the approximation factor of 8β.

The GK improvement derives from two sources. First, it orients each of the concatenated cycles in the direction that minimizes the total latency of the new nodes visited by that cycle. Second, it applies a random shift to the bucket breakpoints. Using buckets of ratio $\gamma \approx 3.59$ instead of ratio 2, it achieves a performance guarantee of $\gamma\beta$.

Our algorithm departs from these previous ones in that we do not start off with approximate k-MSTs for every value of k. Instead, we obtain $(2 - \frac{1}{n-1})$-approximate n_i-MSTs for some subsequence $n_1 < \cdots < n_\ell$ (where $n_1 = 1$ and $n_\ell = n$) that is not under our control. Let d_k denote the cost of the tree spanning k nodes and b_k denote our lower bound on the optimal k-MST for $k = n_1, \ldots, n_\ell$. We derive these trees using a Lagrangian relaxation technique, which allows us to guarantee that linearly interpolating the b_{n_i} to the missing values of k yields valid lower bounds on the cost of the optimal k-MST. We will obtain our MLP solution by concatenating some subset of these trees, as in Blum et al. and GK.

The GK analysis uses the idea of *modified latency*. Roughly, one can think of the modified latency of node v as the average latency of all of the nodes first visited by the cycle in the concatenation that first visits node v. The total modified latency is an upper bound on the latency of the MLP tour we construct. The GK randomized bucketing procedure yields a solution whose expected total modified latency is at most $\gamma(d_2 + \cdots + d_n) \leq \gamma\beta(OPT_2 + \cdots + OPT_n) \leq \gamma\beta OPT$ (where OPT denotes the value of the optimal MLP tour and OPT_k denotes the optimal k-MST value). Goemans and Kleinberg also observe that one can use a shortest path computation to determine which concatenation of trees minimizes the total modified latency. Since the minimum is no more than the expectation, this yields a deterministic $\gamma\beta$-approximation algorithm. Whereas Goemans and Kleinberg introduce the shortest path calculation merely to derandomize their algorithm, for us the use of the shortest path computation is central to the analysis of our performance guarantee.

3. The algorithm. Having motivated and outlined the main points of our MLP algorithm, in this section we describe it precisely.

We start by using our tree-generating algorithm of section 5 to produce some set of ℓ trees $T_{n_1}, \ldots, T_{n_\ell}$ rooted at r and spanning $n_1 < \cdots < n_\ell$ nodes, respectively

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

(where $n_1 = 1$ and $n_\ell = n$). For $k = n_1, n_2, \ldots, n_\ell$, let d_k denote the cost of tree T_k. Without loss of generality, we assume d_k is increasing with k. Our tree-generating algorithm also establishes lower bounds on OPT_k (the cost of the optimal k-MST rooted at r) and hence on the latency of the kth node visited in the optimal MLP tour for $k = 2, \ldots, n$. These lower bounds b_k satisfy the properties:

1. $b_k \le OPT_k$ for $1 \le k \le n$,
2. $d_{n_i} \le \beta b_{n_i}$ for $1 \le i \le \ell$,
3. $b_k = b_{n_{i-1}} + \frac{b_{n_i} - b_{n_{i-1}}}{n_i - n_{i-1}}(k - n_{i-1})$ for $n_{i-1} \le k \le n_i$ and $i = 2, \ldots, n$.

That is, b_k is the linear interpolation of $b_{n_{i-1}}$ and b_{n_i}, and each T_{n_i} is a β-approximate n_i-MST. In our algorithm, we can specify any $\beta \ge (2 - \frac{1}{n-1})$. For now, think of β as 2. Later on, we will choose some β in the interval $(2 - \frac{1}{n-1}, 2)$.

Now we use a shortest path calculation described below to select some subcollection of these trees to concatenate. Denote the selected trees by $T_{j_1}, \ldots, T_{j_m}$, so $j_1, \ldots, j_m$ is the increasing sequence of nodes they span. For each selected tree T_i, double all of the tree edges and traverse an Eulerian tour starting at r, shortcutting nodes already visited, to obtain a cycle $\hat{C}_i$. Now obtain a cycle C_i from $\hat{C}_i$ by shortcutting all nodes (except for r) that are visited by some $\hat{C}_k$, with $k < i$. Because we are in a metric space, this does not increase the length of the cycle. Let S_i denote the set of nodes visited by C_i, excluding the root r. Orient C_i in the direction that minimizes the total latency of the nodes in S_i. To obtain our MLP solution, simply traverse each rooted, oriented cycle $C_{j_1}, \ldots, C_{j_m}$ in order, shortcutting the intermediate visits to the root between cycles. Let $C = C_{j_1}, \ldots, C_{j_m}$ denote this concatenated tour. Following Goemans and Kleinberg, we define the *modified latency* of the kth node of C to be

$$\pi_k = d_{j_{p(k)}} + 2(d_{j_{p(k)-1}} + \cdots + d_{j_1}), \tag{1}$$

where $p(k)$ is the smallest index such that $k \le j_{p(k)}$. The motivation for this definition is that if the sets of nodes spanned by $T_{j_1}, \ldots, T_{j_m}$ are nested, then π_k is an upper bound on the average latency of the nodes first visited by cycle $C_{j_{p(k)}}$. Indeed, in section 4, we repeat an argument of Goemans and Kleinberg that in all cases $\pi_2 + \cdots + \pi_n$ is an upper bound on the total latency of C.

We can now describe the shortest path computation whose solution identifies a tree concatenation with minimum total modified latency. We construct a graph G with nodes $n_1, \ldots, n_\ell$ and arcs $i \to k$ for each $i < k$. A path $j_1 \to \cdots \to j_m$ corresponds to selecting trees $T_{j_1}, \ldots, T_{j_m}$. Thus, the cost on arc $i \to k$ is

$$(k - i)d_k + 2(n - k)d_k = 2d_k \left(n - \frac{i + k}{2} \right), \tag{2}$$

which corresponds to the contribution made to the total modified latency by traversing tree T_k immediately after traversing T_i. This is because tree T_k contributes d_k to the modified latencies of the $(k - i)$ new nodes it visits and $2d_k$ to each of the remaining $(n - k)$ unvisited nodes. If the shortest path from 1 to n in this graph goes $j_1 \to \cdots \to j_m$, then we select trees $T_{j_1}, \ldots, T_{j_m}$ to concatenate.

4. Analyzing the approximation ratio. The analysis of our algorithm proceeds in three steps. First, we demonstrate that $\pi_2 + \cdots + \pi_n$ really is an upper bound on the latency of the tour we construct, even if the trees are not nested. Next, we appeal to a result of Goemans and Kleinberg that upper bounds the total modified latency of the tour given by the shortest path computation in terms of the tree costs, in

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

the event that we have trees of all sizes, $2, \ldots, n$. Finally, we show that, if we were to run this shortest path computation with our real trees and some "phantom" interpolated trees, the computation would never select any of the phantom trees. Therefore, we achieve the same performance guarantee that GK would achieve if we actually did have access to the phantom trees.

For completeness, we begin by repeating an argument of Goemans and Kleinberg showing that the modified latencies do upper bound the latency of the concatenated tour.

LEMMA 4.1 (see [22]). *The total latency of the MLP tour obtained by concatenating trees $T_{j_1}, \ldots, T_{j_m}$ (where $j_1 = 1$, $j_m = n$) is at most $\pi_2 + \cdots + \pi_n$, where the π_k are given by* (1).

Proof. The following argument essentially says that the worst case for our analysis is when the sets of nodes spanned by the selected trees $T_{j_1}, \ldots, T_{j_m}$ are nested. Let us consider the latency of the kth node we visit in the concatenated tour C, where r is considered to be the first node, whose latency is zero. If the kth node in C was encountered as part of cycle C_{j_p}, then we can upper bound its latency by the sum of the costs of cycles $C_{j_1}, \ldots, C_{j_{p-1}}$ plus the portion of cycle C_{j_p} that is traversed prior to reaching this node. Since we traverse cycle C_{j_p} in the direction that minimizes the total latency of the new nodes S_{j_p}, the average contribution of this cycle to the latencies of the nodes in S_{j_p} is at most half the cost of the cycle. To see this, notice that, for any node $i \in S_{j_p}$, traversing C_{j_p} in one direction contributes some amount x to the latency of i, and traversing C_{j_p} in the other direction contributes $\text{cost}(C_{j_p}) - x$, so on average it contributes $\text{cost}(C_{j_p})/2$. The cost of C_i is at most $2d_i$, by the triangle inequality. Therefore, the average latency of the nodes in S_{j_p} is at most $2(d_{j_1} + \cdots + d_{j_{p-1}}) + d_{j_p}$, so the total latency of C is at most

$$\sum_{p=1}^{m} |S_{j_p}|(2(d_{j_1} + \cdots + d_{j_{p-1}}) + d_{j_p}). \tag{3}$$

Since $\sum |S_{j_p}| = n$, we can view this as a weighted sum. Clearly, the worst case for this analysis is when the sets of nodes spanned by the trees $T_{j_1}, \ldots, T_{j_m}$ are nested, since this puts the greatest weight on the larger terms in (3). In this worst case, our upper bound on the average latency of the $(j_{p-1}+1)$th through j_pth nodes in C becomes $2(d_{j_1} + \cdots + d_{j_{p-1}}) + d_{j_p}$. Thus since $\pi_k = d_{j_{p(k)}} + 2(d_{j_{p(k)-1}} + \cdots + d_{j_1})$, where $p(k)$ is the smallest index such that $k \le j_{p(k)}$, our tour has latency at most $\pi_2 + \cdots + \pi_n$. ∎

The advantage of the upper bound $\pi_2 + \cdots + \pi_n$ is that it depends only on the costs of the selected trees and the number of nodes they span, not on the structure of the trees. We now state the main theorem of the Goemans and Kleinberg paper.

DEFINITION 4.2. *The number γ denotes the unique solution of $\gamma \ln \gamma = \gamma + 1$, which is approximately* 3.5912.

THEOREM 4.3 (see [22]). *Given $d_2, \ldots, d_n \ge 0$, let G be the graph on nodes $1, \ldots, n$ including all arcs $i \to k$ for $i < k$, with arc lengths given by* (2). *Then the shortest path in G from node* 1 *to node n has length at most $\gamma(d_2 + \cdots + d_n)$.*

Recall that our tree-generating procedure of section 5 returns trees of sizes $n_1, \ldots, n_\ell$ and costs $d_{n_1}, \ldots, d_{n_\ell}$. It also establishes lower bounds b_k on the cost OPT_k of the optimal k-MST for every k, and these bounds satisfy properties 1–3 from section 3. Let us linearly interpolate the tree costs d_{n_i} to the missing values of k. That is, set

$$d_k = d_{n_{i-1}} + \frac{d_{n_i} - d_{n_{i-1}}}{n_i - n_{i-1}}(k - n_{i-1}) \tag{4}$$

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

for $n_{i-1} \le k \le n_i$ and $i = 2, \ldots, n$. Then clearly $d_k \le \beta b_k$ for all k – it is true for the end points of each interval, so it is true for the linear interpolations. That is, if we actually had a tree of cost d_k spanning k nodes, it would be a β-approximate k-MST, and therefore we could use the GK algorithm to generate a $\beta\gamma$-approximate MLP tour. Unfortunately, we are missing these interpolated trees, so we must find a way to get around this difficulty.

Notice that the GK shortest path computation can be defined independently of trees and MLP tours—it just requires a set of ordered pairs to define the graph G in which the shortest path is computed.

DEFINITION 4.4. *A* tree signature *is an ordered pair (k, c), where k is its* size *and c is its* cost. *If we denote this tree signature by T, then we define $|T| := k$ and $c(T) := c$. Analogously, if T is an actual tree, let $|T|$ denote the number of nodes spanned by it and $c(T)$ denote the total cost of its edges $\sum_{e \in T} c_e$.*

Obviously, a real tree T has an associated tree signature $(|T|, c(T))$. We will find it convenient to consider the behavior of the GK shortest path computation on graphs generated from tree signatures that may or may not correspond to actual trees. When we talk about concatenating a sequence of tree signatures to get a "tour," we just mean to define the kth modified latency π_k with respect to such a concatenation using (1), exactly the same as if these tree signatures represented real trees. Thus, tree signatures, concatenations of tree signatures, and modified latencies are just a bookkeeping mechanism, with the property that if the tree signatures in a concatenation correspond to actual trees, then these trees can be concatenated into an actual MLP tour with latency at most $\pi_2 + \cdots + \pi_n$. When we use a tree signature that may or may not correspond to an actual tree, we may think of it as corresponding to a "phantom tree" that exists only in our minds. Unlike *tree signature*, *phantom tree* is not a technical term but rather just a mnemonic to signify that we *wish* we had a real tree with a particular signature.

In particular, we wish we had a full set of trees with the signatures (k, d_k), $k = 2, \ldots, n$, because this would immediately yield a $\beta\gamma$-approximate MLP tour. Why? Suppose we were to run the GK shortest path computation using the full set of costs $d_2, \ldots, d_n$, i.e., using both the real trees and the phantom trees. Then by Theorem 4.3, the modified latency of the resulting solution would be at most

$$\begin{aligned} \gamma(d_2 + \cdots + d_n) &\le \beta\gamma(b_2 + \cdots + b_n) \\ &\le \beta\gamma(OPT_2 + \cdots + OPT_n) \\ &\le \beta\gamma OPT, \end{aligned}$$

where OPT denotes the optimal MLP value. The difficulty is that the shortest path computation might select one of the phantom trees, in which case we cannot actually construct the MLP tour. Fortunately, we will show that this never occurs, because the shortest path will go only through "corner points."

DEFINITION 4.5. *With respect to a set of tree signatures $(1, 0), (2, c_2), \ldots, (n, c_n)$, the signature (k, c_k) is a* corner point *if $k \in \{1, n\}$ or $k \in \{2, \ldots, n-1\}$ and $c_k \ne \frac{1}{2}(c_{k-1} + c_{k+1})$.*

The corner points divide up the interval $\{1, \ldots, n\}$ such that the tree signatures between any two consecutive corner points are just linear interpolations of those two corner points.

THEOREM 4.6. *The Goemans–Kleinberg shortest path computation in the graph generated from a set of tree signatures $(1, 0), (2, c_2), \ldots, (n, c_n)$ visits only corner points (with respect to these signatures).*

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

Proof. Just to eliminate special cases that we would otherwise have to consider, let us introduce self-loops at every node in our graph, with costs still given by (2). In any case in our argument where we end up using a self-loop, we can obtain an even shorter path by removing it.

Suppose on the contrary that a shortest path visits $i \to j \to k$, where j is not a corner point. Let n_{lo} and n_{hi} be the sizes of the two nearest corner points that sandwich j, where $n_{\text{lo}} < j < n_{\text{hi}}$. Set $\lambda = (c_{n_{\text{hi}}} - c_{n_{\text{lo}}})/(n_{\text{hi}} - n_{\text{lo}})$; i.e., λ is the slope of the line connecting these two corner points. If $\lambda \leq 0$, we can decrease the modified latencies $\pi_{i+1}, \ldots, \pi_n$ by visiting $j+1$ instead of j. (This corresponds to replacing a tree in the concatenation by a larger one of lesser cost.) Thus, we know $\lambda > 0$.

Treating j as a variable now, we show that we can obtain a strictly shorter path by setting j to either $\max(i, n_{\text{lo}})$ or $\min(k, n_{\text{hi}})$, arriving at a contradiction. For j in the interval between the sandwiching corner points, the tree signature (j, c_j) is just the linear interpolation of these corner points, i.e., $c_j = c_{n_{\text{lo}}} + \lambda(j - n_{\text{lo}})$. Thus, by definition of the arc lengths (2), the subpath from $i \to j \to k$ costs

$$2(c_{n_{\text{lo}}} + \lambda(j - n_{\text{lo}})) \left(n - \frac{i+j}{2}\right) + 2c_k \left(n - \frac{j+k}{2}\right). \tag{5}$$

This cost is valid for $\max(i, n_{\text{lo}}) \leq j \leq \min(k, n_{\text{hi}})$ and is a quadratic function of j, where the coefficient on the j^2 term is $-\lambda$. Thus, the cost is strictly concave in j, so it attains a strict minimum at one of the end points $\max(i, n_{\text{lo}})$ or $\min(k, n_{\text{hi}})$. This is a contradiction, because we already started with a shortest path. ∎

We chose phantom trees that were linear interpolations of the real trees we had. Thus, all of the corner points in our set of tree signatures correspond to real trees. Since a shortest path in the graph with the phantom trees included never actually uses any of the phantom trees, we might as well run the shortest path computation using just the actual trees, as in the algorithm description of section 3. Putting Theorem 4.6 together with the discussion preceding it yields our main result. Recall that $\beta < 2$.

THEOREM 4.7. *The algorithm described in section* 3 *yields an MLP tour of cost at most* $\beta\gamma OPT$.

5. Generating trees and lower bounds via Lagrangian relaxation. In this section, we discuss how to use Lagrangian relaxation to obtain trees and lower bounds satisfying properties 1–3 of section 3.

Lagrangian relaxation is a technique for generating lower bounds on the optimal solution of one minimization problem by relaxing some of the constraints to generate a related but simpler minimization problem, which ideally should be easier to solve. We discuss here how the PCST problem serves as a Lagrangian relaxation of the k-MST problem, a relationship first explored by [13].

Given a graph with a cost c_e on each edge e, a root node r, and a penalty $p_v \geq 0$ for each node v, the PCST problem asks for a tree T rooted at r that minimizes the quantity

$$\sum_{e \in T} c_e + \sum_{v \notin T} p_v.$$

In other words, we ask for a tree that minimizes the sum of the edge costs in the tree plus the penalties of the nodes not spanned by the tree. Throughout this paper, we use PCST to refer to the PCST problem with *uniform penalties*, where all of the

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

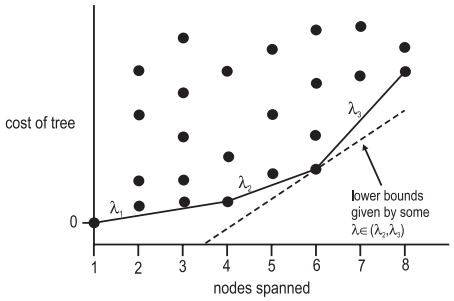

FIG. 2. *The dots represent a signature* $(|T|, c(T))$ *for each possible tree* T *rooted at* r. *The lowest dot in column* k *represents the optimal* k*-MST, while the lines forming the lower convex hull of the dots represent the best lower bounds on* OPT_k *that could be derived via our Lagrangian relaxation.*

penalties are set to the same value $\lambda \geq 0$. In this case, the objective function becomes

$$\sum_{e \in T} c_e + \lambda(n - |T|).$$

We now consider how the PCST problem can generate lower bounds for the k-MST problem. Let $T^*(\lambda)$ be an optimal solution to the PCST problem when the penalty parameter is set to λ. Since all trees spanning $|T^*(\lambda)|$ nodes incur the same penalty term, $T^*(\lambda)$ must be the optimal $|T^*(\lambda)|$-MST. Moreover, since the optimal k-MST is one feasible solution to the PCST problem, we have $c(T^*(\lambda)) + \lambda(n - |T^*(\lambda)|) \leq OPT_k + \lambda(n - k)$. Rearranging gives

$$c(T^*(\lambda)) + \lambda(k - |T^*(\lambda)|) \leq OPT_k. \tag{6}$$

Notice that solving a *single* instance of PCST generates k-MST lower bounds for *all* values of k simultaneously. Since the PCST problem is also NP-hard, this does not help us directly in generating trees and k-MST lower bounds. However, we will continue with this thought experiment, which will guide us in how we use the Goemans–Williamson approximation algorithm for PCST.

The following graphical interpretation of this Lagrangian relaxation is revealing. Consider a plot of the signatures $(|T|, c(T))$ for all possible trees T rooted at r (see Figure 2). Let $\mathcal{T}$ denote this set of points. Fixing the penalty parameter $\lambda \geq 0$, consider a line of slope λ and slide this line up vertically until the first time it hits one of the points in $\mathcal{T}$. This is the point corresponding to $T^*(\lambda)$, and the line $\mathcal{L}(\lambda)$ of slope λ passing through it represents the k-MST lower bounds generated for all values of k. Thus, if we consider the best lower bounds that can be generated by solving PCST for all possible values of λ, this just yields the lower convex hull of the point set $\mathcal{T}$. Let $P_0, \ldots, P_\ell$ denote the sequence of tree signatures along the lower convex hull of $\mathcal{T}$ starting with $P_0 = (1, 0)$ and t_i denote the number of nodes spanned by the tree corresponding to P_i. Let $\lambda_0 = 0$, and for $i = 1, \ldots, \ell$ let λ_i be the slope of the line connecting P_{i-1} to P_i. Consider how the line $\mathcal{L}(\lambda)$ moves as we change λ. As λ increases from 0 to λ_1, the line starts out horizontally and pivots around the point P_0 until it hits P_1. In general, as λ increases from λ_i to λ_{i+1}, the line $\mathcal{L}(\lambda)$ rotates around point P_i until it hits point P_{i+1}. Now consider how the k-MST lower bound changes with λ, for various values of k. If $k = t_i$, then the lower bound on OPT_k strictly increases for $\lambda \in [0, \lambda_i)$, stays constant on $[\lambda_i, \lambda_{i+1}]$, and then decreases on (λ_{i+1}, ∞).

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

If $k \in (t_{i-1}, t_i)$, then the lower bound increases on $[0, \lambda_i)$, attains its maximum at λ_i, and decreases on (λ_i, ∞). We refer to the values λ_i $(i = 1, \ldots, \ell)$, where the size of the optimal tree changes, as *critical values* with respect to the optimal algorithm.

In general, suppose $\mathcal{A}$ is any PCST algorithm such that the tree generated (as a function of λ) changes only at a finite number of breakpoints and remains constant on the intervals between consecutive breakpoints. Then we say that $\mathcal{A}$ has *critical values* and refer to those breakpoints as *critical values with respect to the algorithm* $\mathcal{A}$. If the algorithm is understood, we will just call them critical values.

Notice that, if we want to generate the best lower bounds for all values of k and also the trees corresponding to the points P_i, it is sufficient to solve the PCST problem just for the critical values of λ. The critical value λ_i generates the best k-MST lower bounds for k in the interval $[t_{i-1}, t_i]$. By using all of the critical values, this collection of intervals covers all values of k in $[1, n]$. Notice that this collection of trees and lower bounds would satisfy conditions 1–3 of section 3, with $\beta = 1$.

Since the PCST problem is NP-hard, we cannot expect to solve it to optimality in polynomial time. Fortunately, Goemans and Williamson [21] gave a $(2 - \frac{1}{n-1})$-approximation algorithm that also has the *Lagrangian multiplier preserving* property. This means that running the algorithm with penalty parameter λ returns a tree $T(\lambda)$ and a lower bound $LB(\lambda)$ such that

$$c(T(\lambda)) \leq \left(2 - \frac{1}{n-1}\right) LB(\lambda) \tag{7}$$

and

$$OPT_k \geq LB(\lambda) + \lambda(k - |T(\lambda)|). \tag{8}$$

In other words, the tree generated is a $(2 - \frac{1}{n-1})$-approximate $|T(\lambda)|$-MST, and the line $\mathcal{L}(\lambda)$ of slope λ through $(|T(\lambda)|, LB(\lambda))$ gives valid k-MST lower bounds for all values of k. We will prove these inequalities in Lemma 9.2 of section 9.

How do these lower bounds change as we vary λ? The inner workings of the Goemans–Williamson PCST algorithm are such that the algorithm has critical values.[2] At a critical value of λ, we can generate the tree corresponding to one bordering interval or the other depending on how we break ties in the algorithm, which does not affect the line of lower bounds $\mathcal{L}(\lambda)$. Thus, if the two trees are T_{lo} and T_{hi} with corresponding lower bounds $LB_{T_{\text{lo}}}$ and $LB_{T_{\text{hi}}}$ (where $|T_{\text{lo}}| < |T_{\text{hi}}|$), then the interpolated k-MST lower bounds for $|T_{\text{lo}}| < k < |T_{\text{hi}}|$ coincide with the line $\mathcal{L}(\lambda)$ and are hence valid lower bounds. We say that this critical value of λ *covers the size interval* $[|T_{\text{lo}}|, |T_{\text{hi}}|]$ and *covers the bound interval* $[LB_{T_{\text{lo}}}, LB_{T_{\text{hi}}}]$. We refer to both the size interval and the bound interval as *critical intervals*.

Suppose that we can find a collection of critical values of λ such that the union of the size intervals they cover contains $\{1, \ldots, n\}$. Each critical value generates a line of lower bounds for the size interval that it covers. Let $\mathcal{T}$ denote the set of lower bound tree signatures generated by all of these critical values. If we take the lower convex hull of $\mathcal{T}$, the interpolated lower bounds would be at most the lower bounds generated by the covering intervals, and hence still valid (see Figure 3). Thus, the desired properties 1–3 of section 3 would hold, with $\beta = 2 - \frac{1}{n-1}$.

How can we find the required critical values of λ? Let $c_{\max}$ denote the distance from the root to the farthest node. Because of the inner workings of the PCST

[2] Unlike in the case of $T^*(\lambda)$, on the intervals where $T(\lambda)$ is constant, $LB(\lambda)$ may not be constant, and $|T(\lambda)|$ is not increasing in λ. However, this does not cause a problem.

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

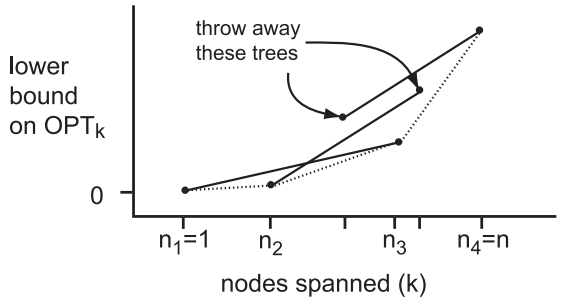

FIG. 3. *Each dot represents a (size of tree, lower bound) pair returned by PCST, and the solid lines are the interpolated lower bounds from Lemma* 9.2. *The lower envelope (dotted line) is still clearly a valid lower bound at each point, since it is below the solid lines. We keep only the trees whose dots are on the lower envelope.*

algorithm, it is a fact that $T(c_{\max})$ spans all n nodes, while $T(0)$ spans only the root. For any arbitrary target value $k \in \{1, \ldots, n\}$, there is at least one critical value λ^* that covers a size interval containing k. Moreover, if we maintain an interval $[\lambda_{\text{lo}}, \lambda_{\text{hi}}]$ such that $|T(\lambda_{\text{lo}})| \leq k \leq |T(\lambda_{\text{hi}})|$, we know that there is some solution $\lambda^* \in [\lambda_{\text{lo}}, \lambda_{\text{hi}}]$. We can initialize the interval to $[0, c_{\max}]$, test some value λ in the interval, and update the appropriate end point to maintain the invariant. It is a fact that all of the decisions in the PCST algorithm involve comparing quantities that are linear in λ, and therefore we can apply the parametric search technique of Megiddo to calculate λ^* exactly [29]. This requires running the PCST algorithm once for each comparison made in the PCST algorithm. Using the $O(n^2)$ implementation of Gabow and Pettie [18], this means that we can compute a single critical value exactly using $O(n^2)$ PCST calls. Since we need to compute at most $(n-1)$ critical values to cover $[1, n]$, we obtain the following theorem.

THEOREM 5.1. *There is a* $(2 - \frac{1}{n-1})\gamma$*-approximation algorithm for the minimum latency problem that runs in time dominated by* $O(n^3)$ *PCST calls.*

Aside from justifying (7) and (8) (which we do in section 9), this ends the discussion of our basic algorithm for the MLP problem. In section 6, we discuss how to speed up this running time by allowing some small error in our computation of a critical value λ^*, and, in section 7, we discuss how to further speed up the algorithm by using randomization to reduce the number of critical values we need to calculate. Before we move on to this, let us describe a slight variant of our algorithm.

In section 3, we discussed creating tree signatures by interpolating the sizes and costs of pairs of real trees. We now describe a slightly different algorithm, based on creating tree signatures from the interpolated lower bounds rather than interpolating real trees. This alternate view will be useful in section 7 when we discuss our randomized algorithm.

Let $b_1, \ldots, b_n$ be the lower bounds given by the lower convex hull of $\mathcal{T}$, described above. Suppose we construct the graph G of section 3 using the tree signatures (k, b_k), $k = 1, \ldots, n$, and compute the shortest path to select which tree signatures to concatenate. By Theorem 4.3, the shortest path will have modified latency at most $\gamma(b_2 + \cdots + b_n) \leq \gamma OPT$, and, by Theorem 4.6, it will use only corner points. Since every corner point is a tree signature corresponding to the end point of an interval we generated, we may replace these phantom trees with real trees, blowing up the tree costs and hence the modified latencies by at most a factor of β. Thus, our tour has total latency at most $\beta\gamma OPT$.

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

6. Accelerating the running time. The goal of this section is to reduce the number of PCST calls required by our algorithm to $O(n \log n)$ while achieving an approximation ratio of 2γ.

Because the PCST algorithm gives an approximation factor of $(2 - \frac{1}{n-1})$ instead of 2, this allows us a little bit of slack if we just want a 2γ-approximation algorithm for the MLP problem. We can use this slack to significantly speed up our algorithm. Instead of using Megiddo's parametric search technique to calculate a critical value λ^* exactly, we can instead use binary search to find a very small interval $[\lambda_{\text{lo}}, \lambda_{\text{hi}}]$ that contains λ^*. If the interval is small enough, then the lines $\mathcal{L}(\lambda_{\text{lo}})$ and $\mathcal{L}(\lambda_{\text{hi}})$ are close enough together that we can replace them with a single line that gives valid lower bounds b_k on the interval $[|T(\lambda_{\text{lo}})|, |T(\lambda_{\text{hi}})|]$ without degrading the lower bounds by much.

By a *probe*, we mean a single PCST call to narrow the interval $[\lambda_{\text{lo}}, \lambda_{\text{hi}}]$. By a *critical search*, we mean a series of probes used to narrow the interval $[\lambda_{\text{lo}}, \lambda_{\text{hi}}]$ to the desired size by binary search, so that the following three conditions hold:

1. $b_k \leq OPT_k$ for $k_{\text{lo}} \leq k \leq k_{\text{hi}}$,
2. $c(T(\lambda)) \leq \beta b_{|T(\lambda)|}$ for $\lambda \in \{\lambda_{\text{lo}}, \lambda_{\text{hi}}\}$,
3. $b_k = b_{k_{\text{lo}}} + \frac{b_{k_{\text{hi}}} - b_{k_{\text{lo}}}}{k_{\text{hi}} - k_{\text{lo}}}(k - k_{\text{lo}})$,

where $k_{\text{lo}} = |T(\lambda_{\text{lo}})|$ and $k_{\text{hi}} = |T(\lambda_{\text{hi}})|$. The bounds $b_{k_{\text{lo}}}$ and $b_{k_{\text{hi}}}$ will take the form $\delta \text{LB}(\lambda_{\text{lo}})$ and $\delta \text{LB}(\lambda_{\text{hi}})$, for some δ less than but close to 1 that we can specify in the algorithm, and $\beta = \frac{1}{\delta}(2 - \frac{1}{n-1})$.

Let $c_{\min}$ denote the distance from the root to the closest other node. In Corollary 9.5 of section 9, we show that $O(\log \frac{n c_{\max}}{c_{\min}})$ probes are sufficient to perform a critical search with $\beta = (2 - \frac{1}{2n})$. We now show how to reduce the problem to the case where $\frac{c_{\max}}{c_{\min}} = \text{poly}(n)$, so each critical search takes only $O(\log n)$ probes. Since we require $O(n)$ critical searches, our overall algorithm takes at most $O(n \log n)$ PCST calls, as advertised.

How do we ensure $\frac{c_{\max}}{c_{\min}} = \text{poly}(n)$? We first delete every node that is within distance $\frac{c_{\max}}{4n^3}$ of the root (aside from the root itself). Clearly, this did not increase the cost of an optimal solution. We are left with an instance where $\frac{c_{\max}}{c_{\min}} \leq 4n^3$, and we apply our main algorithm to this instance, with $\beta = (2 - \frac{1}{2n})$. This yields a tour of latency at most $(2 - \frac{1}{2n})\gamma OPT$. To obtain a tour for the original instance, we first visit the deleted nodes in arbitrary order, return to the root, and then visit the remaining nodes in the order given by our main algorithm. By the triangle inequality, the total length of the initial partial tour is at most $\frac{c_{\max}}{2n^2}$. By visiting all of these points first, it contributes at most an extra $\frac{c_{\max}}{2n}$ to the total latency of the entire tour. Since $c_{\max} \leq OPT_n \leq OPT$, the total latency of our tour is at most $(2 - \frac{1}{2n})\gamma OPT + \frac{c_{\max}}{2n} < 2\gamma OPT$. This yields the following theorem.

THEOREM 6.1. *The algorithm described above is a 2γ-approximation algorithm for the minimum latency problem and runs in time dominated by $O(n \log n)$ PCST calls.*

7. A faster randomized algorithm. In this section, we describe a randomized algorithm that achieves the same approximation ratio of 2γ (in expectation) while reducing the number of PCST calls to $O(\log^2 n)$. We also show how to derandomize it to achieve a deterministic algorithm with the improved running time but a slightly worse approximation guarantee.

In the paper of Goemans and Kleinberg [22], Theorem 4.3 is proven via the probabilistic method. They actually construct a random path in their graph and show that the expected modified latency of their path is at most $\gamma(d_2 + \cdots + d_n)$.

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

Therefore, the shortest path is at least this short, which yields Theorem 4.3. Thus, instead of Theorem 4.3, a more fundamental statement of their main result is the following.

THEOREM 7.1 (see [22]). *Given $d_2, \ldots, d_n \geq 0$, consider the graph G on nodes $1, \ldots, n$ including all arcs $i \to k$ for $i < k$, with arc lengths given by (2). Let the random variable U be drawn uniformly from $[0, 1)$, let $L_0 = \gamma^U$, and for each integer i define $L_i = L_0\gamma^i$. For each i, mark the largest node j such that $d_j \leq L_i$. Then the expected length of the path from 1 through all of the marked nodes (in order) to n is at most $\gamma(d_2 + \cdots + d_n)$.*

We include a proof here, since we will need to generalize this result for our purposes.

Proof. Recall that the path in G represents a concatenation of tree signatures, and the length of the path equals the sum of the modified latencies $\pi_2 + \cdots + \pi_n$. Fix $k \in \{2, \ldots, n\}$. Let $\mathcal{S} = \{L_0\gamma^i : i \in \mathbb{Z}\}$ be the set of thresholds, and let $L = \min\{L' \in \mathcal{S} : L' \geq d_k\}$. Then L has the same distribution as $d_k\gamma^U$. Moreover, the first tree signature in the concatenation with size at least k has cost at most L, and the smaller tree signatures have costs at most $L\gamma^{-i}$ for $i = 1, 2, \ldots$. Thus,

$$\pi_k \leq L + 2L(\gamma^{-1} + \gamma^{-2} + \cdots) = \frac{\gamma + 1}{\gamma - 1}L,$$

$$E[\pi_k] \leq \frac{\gamma + 1}{\gamma - 1}E[d_k\gamma^U] = \frac{\gamma + 1}{\ln \gamma}d_k = \gamma d_k.$$

The last equality comes because γ satisfies the identity $\gamma \ln \gamma = \gamma + 1$ (from Definition 4.2). Summing over k, the theorem follows by linearity of expectation. ∎

This result generalizes readily to the following.

THEOREM 7.2. *Given $d_2, \ldots, d_n \geq 0$, select random thresholds as in Theorem 7.1. For each threshold L, select one arbitrary tree signature $T(L)$ such that $|T(L)| \geq \max\{k : d_k \leq L\}$ and $c(T(L)) \leq L$. The concatenation of these tree signatures has total modified latency at most $\gamma(d_2 + \cdots + d_n)$, in expectation.*

Proof. Fix $k \in \{2, \ldots, n\}$, and let L be the smallest threshold such that $d_k \leq L$. By assumption, $|T(L)| \geq k$. Then by the time the concatenation includes tree signature $T(L)$, it has visited at least k nodes, and the sum of the costs of all previous tree signatures in the concatenation is at most $L(\gamma^{-1} + \gamma^{-2} + \cdots)$. Thus, $\pi_k \leq L + 2L(\gamma^{-1} + \gamma^{-2} + \cdots)$, and the rest follows as in the proof of Theorem 7.1. ∎

We will use this theorem to analyze the following randomized algorithm, which reduces the number of critical searches to $O(\log n)$ down from the $O(n)$ of section 6.

Our algorithm starts by generating random thresholds, as in Theorem 7.2. For each threshold L in the range $[c_{\min}, nc_{\max}]$, we perform a critical search with $\beta = 2 - \frac{1}{2n}$ to cover a bound range that includes L. Each of these $O(\log \frac{nc_{\max}}{c_{\min}})$ critical searches generates two actual trees. We use this collection of trees to create the graph G of section 3 and compute the shortest path to determine which of these trees to concatenate. In order to limit the number of critical searches to $O(\log n)$, we preprocess the original input by deleting every node that is within distance $\frac{c_{\max}}{8n^3}$ of the root (aside from the root itself). We visit all of these nodes first in arbitrary order and then follow the tour given by the tree concatenation. The reason that we consider only thresholds in the range $[c_{\min}, nc_{\max}]$ is that $c_{\min} \leq OPT_2$ and $OPT_n \leq nc_{\max}$, so these thresholds are the only relevant ones.

The tree signatures $T(L)$ mentioned in Theorem 7.2 are used only for the analysis and not for the actual algorithm. They will be derived from the critical searches.

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

THEOREM 7.3. *The randomized algorithm described above yields a 2γ-approximate solution to the minimum latency problem in expectation and runs in time dominated by $O(\log^2 n)$ PCST calls.*

Proof. The number of thresholds we must consider is $O(\log \frac{nc_{\max}}{c_{\min}}) = O(\log n)$, since we preprocessed the input to ensure $\frac{c_{\max}}{c_{\min}} = \text{poly}(n)$. For each of these, we perform one critical search with $\beta = 2 - \frac{1}{2n}$, each of which requires only $O(\log \frac{nc_{\max}}{c_{\min}}) = O(\log n)$ PCST calls, as we will show in Corollary 9.5. Thus, the total number of PCST calls is $O(\log^2 n)$.

Let us analyze the approximation ratio we achieve on the amended instance. For each threshold L, select one tree signature as follows. Our critical search for L covers some size interval $[k_{\text{lo}}, k_{\text{hi}}]$ and some bound interval $[b_{k_{\text{lo}}}, b_{k_{\text{hi}}}]$ containing L. Thus, there is some $k \in \{k_{\text{lo}}, k_{\text{lo}} + 1, \ldots, k_{\text{hi}} - 1\}$ such that the interpolated bounds satisfy $b_k \le L < b_{k+1}$. Let $T(L)$ be a tree signature of size k and cost b_k. Because b_{k+1} is a valid lower bound on OPT_{k+1}, we get $\max\{j : OPT_j \le L\} \le k = |T(L)|$. Thus, this choice of tree signatures satisfies the hypotheses of Theorem 7.2. Let π denote the total modified latency given by concatenating this collection of tree signatures. Theorem 7.2 gives $E[\pi] \le \gamma(OPT_2 + \cdots + OPT_k) \le \gamma OPT$. Now we need to understand how π compares to the cost of the MLP solution we generated.

Recall that each critical search covers some size interval $[k_{\text{lo}}, k_{\text{hi}}]$ and generates a set of lower bounds on OPT_k for k in this size interval. We can think of each of these lower bounds as a tree signature. Consider taking the lower convex hull of all of these tree signatures generated by all of our critical searches, as in Figure 3. For each tree signature $T(L)$ chosen in the last paragraph, replace it with a tree signature of the same size but cost equal to the corresponding point on the lower convex hull. This can only decrease the total modified latency. By Theorem 4.6, we can further improve the modified latency by selecting the best concatenation using only tree signatures corresponding to the corners of the convex hull. But these tree signatures correspond to end points of the size intervals covered by the critical searches, and therefore we can replace these phantom trees with real trees of cost at most β times as much, which were generated by our critical searches. This increases the modified latency to at most $\beta\pi$. But the modified latency of the concatenation we selected is no larger than this, since we selected the best concatenation from a (perhaps) larger set of trees corresponding to all end points of size intervals covered by critical searches, not just the ones corresponding to the lower convex hull of the lower bounds. Thus, the expected latency of our concatenation is at most $E[\beta\pi] \le \beta\gamma OPT$. All of the nodes within distance $\frac{c_{\max}}{8n^3}$ of the root were visited in arbitrary order first, which increases the total latency of the tour by at most $\frac{c_{\max}}{4n} \le \frac{OPT_n}{4n} \le \frac{OPT}{4n}$. Thus, the final tour has expected latency at most $(2 - \frac{1}{4n})\gamma OPT < 2\gamma OPT$. ∎

As Goemans and Kleinberg observed, the algorithm above can be derandomized using the following observation. Instead of choosing $L_0 = \gamma^U$, where U is drawn uniformly from $[0, 1)$, fix some positive integer p, and draw U uniformly from $\{\frac{k}{p} : 0 \le k < p\}$. This changes the factor of γ in the approximation guarantee of Theorem 7.2 to

$$\frac{(\gamma + 1)\gamma^{\frac{1}{p}}}{p(\gamma^{\frac{1}{p}} - 1)},$$

which we denote by $r(p)$. Therefore, if we run the randomized algorithm for each of the p choices of the discrete random variable U and take the best solution, we get a deterministic algorithm that achieves an approximation factor of $\beta r(p)$, plus the tiny term for the nodes we deleted to create the amended instance.

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

As we would expect, $r(p) \to \gamma$ as $p \to \infty$. A Taylor series expansion reveals that

$$r(p) = \gamma\left(1 + \frac{\ln \gamma}{2p} + O\left(\frac{1}{p^2}\right)\right).$$

This yields the following result.

THEOREM 7.4. *For any $\epsilon > 0$, there is a deterministic algorithm for the minimum latency problem that gives an approximation factor of $(2 - \frac{1}{4n})\gamma(1+\epsilon)$ and runs in time dominated by $O(\frac{1}{\epsilon}\log^2 n)$ PCST calls.*

Setting $\epsilon = \frac{1}{8n}$, we can recover a deterministic 2γ-approximation algorithm using $O(n \log^2 n)$ PCST calls, which is slower than our old deterministic algorithm. But for any constant $\epsilon > 0$, we save a factor of $\frac{n}{\log n}$ compared to Theorem 6.1.

8. The weighted minimum latency problem. In this section, we consider the weighted MLP, in which each node v has an associated positive integer weight w_v. The goal is to find a tour that minimizes the sum over all nodes of the weight of the node times its latency. As in the unweighted case, the latency of a node in a tour is its distance along the tour from the root node. Since the root node r always has zero latency, its weight is irrelevant. The unweighted case discussed so far is simply the case in which $w_v = 1$ for all nodes v.

8.1. A pseudopolynomial-time algorithm. As we observed in section 1, from a weighted instance we can derive an equivalent unweighted instance by replacing each node v with a clique of w_v nodes at distance zero from each other. This new instance has $W = \sum_v w_v$ nodes. Recall that all weights are positive integers, so $W \geq n$. We can apply our deterministic algorithm of section 3 to obtain a 2γ-approximation while making $O(W \log W)$ PCST calls. In particular, we use our tree-generating algorithm of section 5 (along with the tweaks of section 6 to speed up the running time) to find some set of ℓ trees $T_{t_1}, \ldots, T_{t_\ell}$ rooted at r and spanning total weights $t_1 < \cdots < t_\ell$ respectively, where $t_1 = 1$ and $t_\ell = W$. For $i = t_1, t_2, \ldots, t_\ell$, we let d_i denote the cost of tree T_i. The tree-generating algorithm also establishes lower bounds b_w on OPT_w, the cost of the optimal tree rooted at r that has total node weight w. We call this a w-MST, as it generalizes the notion of a k-MST. Just as the cost of an optimal k-MST gives a lower bound on the latency of the kth node visited in the optimal MLP tour, the cost OPT_w of an optimal w-MST gives a lower bound on the length of the path in the optimal MLP tour from the root until nodes of total weight at least w have been visited. We claim then that $\sum_{w=1}^{W} OPT_w$ gives a lower bound on the cost of any weighted MLP tour. To see this, observe that, if l_i is the latency of the ith node visited in an optimal MLP tour, w_i is the weight of that node, and $W_i = \sum_{j=1}^{i} w_j$, then the cost of the optimal MLP tour is

$$\sum_{i=1}^{n} w_i l_i \geq \sum_{i=1}^{n} w_i OPT_{W_i} \geq \sum_{i=1}^{n} \sum_{w=W_{i-1}+1}^{W_i} OPT_w = \sum_{w=1}^{W} OPT_w.$$

As in the unweighted case, our lower bounds satisfy the properties that

1. $b_w \leq OPT_w$ for $1 \leq w \leq W$,
2. $d_{t_i} \leq \beta b_{t_i}$ for $1 \leq i \leq \ell$,
3. $b_w = b_{t_{i-1}} + \frac{b_{t_i} - b_{t_{i-1}}}{t_i - t_{i-1}}(w - t_{i-1})$ for $t_{i-1} \leq w \leq t_i$ and $i = 2, \ldots, \ell$

for some $\beta > 2 - \frac{1}{n-1}$ that we will specify in the algorithm. We will show in section 9 that we can run the PCST algorithm on the weighted instance without creating

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

extra nodes, so that each invocation of the PCST algorithm runs in $O(n^2)$ time. The algorithm and analysis of section 6 carry over, so we obtain a 2γ-approximation algorithm for the weighted MLP that uses $O(W \log W)$ PCST calls and takes overall time $O(\max(n^2 W \log W, W^2))$. The max comes because the shortest path computation is on a directed acyclic graph with W nodes and $\Theta(W^2)$ edges. This might dominate the running time of the PCST calls if W is large.

8.2. Weakly polynomial-time algorithms. In section 7, we converted our deterministic algorithm for the unweighted MLP to a faster randomized version. The exact same method applies to the weighted MLP. The only difference is in which nodes we delete to create the amended instance. This time we delete all nodes that are within distance $\frac{c_{\max}}{8n^2W}$ of the root, so $\frac{c_{\max}}{c_{\min}} \le 8n^2W$ in the amended instance. We still set $\beta = 2 - \frac{1}{2n}$ when we perform our critical searches. By Corollary 9.5, each one requires $O(\log \frac{Wc_{\max}}{c_{\min}}) = O(\log(nW)) = O(\log W)$ PCST calls (since $W \ge n$). The number of relevant threshold values is $O(\log \frac{nc_{\max}}{c_{\min}}) = O(\log W)$. At the end, our shortest path computation is on a directed acyclic graph with $O(\log^2 W)$ edges, so the running time is dominated by the $O(\log^2 W)$ PCST calls. The weighted latency of our tour on the amended instance is at most $(2 - \frac{1}{2n})\gamma OPT$, in expectation. The at most n nodes that were deleted to create the amended instance can be visited in arbitrary order at the beginning of the tour, which adds at most $\frac{c_{\max}}{4nW}$ to the latency of each node and so at most $\frac{c_{\max}}{4n} \le \frac{OPT_W}{4n} \le \frac{OPT}{4n}$ to the total latency of the tour.

THEOREM 8.1. *The randomized algorithm described above yields an approximation guarantee of 2γ in expectation for the weighted minimum latency problem and runs in time dominated by $O(\log^2 W)$ PCST calls.*

The algorithm above actually gives an approximation factor of less than $2(1 - \frac{1}{8n})\gamma$. Thus, if we derandomize it as in section 7, choosing p large enough so that $r(p) \le 1 + \frac{1}{8n}$, we get a deterministic algorithm with a factor of at most 2γ. If we only want a factor of $2\gamma(1+\epsilon)$, we can save on the running time.

COROLLARY 8.2. *The deterministic algorithm described above for the weighted MLP gives an approximation factor of 2γ using $O(n \log^2 W)$ PCST calls, or a factor of $2\gamma(1+\epsilon)$ using only $O(\frac{1}{\epsilon}\log^2 W)$ PCST calls, for any $\epsilon > 0$.*

Proof. As in Theorem 7.4, setting $p = O(n)$ is sufficient to make $r(p) \le 1 + \frac{1}{8n}$, and setting $p = O(\frac{1}{\epsilon})$ is sufficient to make $r(p) \le (1+\epsilon)$. ∎

8.3. Strongly polynomial-time algorithms. Using one more clever trick, we can make the running time depend only on n and not on W. As in section 8.2, we do this first for our randomized algorithm, and then we derandomize it.

Corollary 9.5 of section 9 will show that we can perform each critical search using at most $O(\log \frac{nWc_{\max}}{w_{\min}c_{\min}})$ probes (i.e., PCST calls), where $w_{\min} = \min_{v \ne r} w_v$. Again, the number of relevant threshold values in the randomized algorithm of section 7 is $O(\log \frac{nc_{\max}}{c_{\min}})$. Thus, if we can limit ourselves to solving only amended instances where both $\frac{W}{w_{\min}}$ and $\frac{c_{\max}}{c_{\min}}$ are $\exp(\text{poly}(n))$, then we will achieve a strongly polynomial running time. In our weakly polynomial algorithm of section 8.2, we had $\frac{c_{\max}}{c_{\min}} = O(W\text{poly}(n))$, which yielded a dependence on $\log W$ in the running time.

We have already taken advantage of the fact that we can first visit all of the nodes very close to the root without increasing the total latency by much. Since we need to decrease $\frac{c_{\max}}{c_{\min}}$, we will need to refine this reasoning. Since the weighted latency of the node sitting at distance $c_{\max}$ from the root is at least $w_{\min}c_{\max}$, this is a lower bound on OPT. Thus, if we visit every node that is within distance $\frac{c_{\max}w_{\min}}{8n^2W}$ of the root first, this contributes at most $\frac{OPT}{4nW}$ to the latency of each node and hence at most $\frac{OPT}{4n}$

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

to the total weighted latency of the tour. Thus, we can amend our instance to one in which $\frac{c_{\max}}{c_{\min}} \leq \frac{8n^2W}{w_{\min}}$ while giving up only an extra $\frac{1}{4n}$ in the approximation factor. Therefore, we need only to worry about making $\frac{W}{w_{\min}}$ small. We will accomplish this by visiting the nodes of very small weight at the end.

With this motivation behind us, here is the description of our algorithm. Sort the nonroot nodes by lowest to highest weight. Scan the sorted list from the beginning, and insert a dividing point between any two consecutive nodes for which the ratio of their weights is more than $8n^3$. Let B_i denote the set of nodes between the $(i-1)$th and ith dividing points. We will call this set a *block*, and we will construct subtours on each block independently. If the blocks are $B_1, B_2, \ldots, B_\ell$, we will traverse their subtours in the order $B_\ell, B_{\ell-1}, \ldots, B_1$. Within each block, we will apply the randomized algorithm of section 8.2, where the amended instance for each block is derived by deleting all nodes within distance $\frac{c'_{\max} w'_{\min}}{8n^2 W'}$ of the root, where $w'_{\min}$ and W' are, respectively, the minimum weight of any node and the total weight of all nodes in the block.

THEOREM 8.3. *The randomized algorithm described above for the weighted MLP yields an approximation factor of* $(2-\frac{1}{4n})\gamma$ *in expectation and runs in time dominated by* $O(n^2 \log^2 n)$ *PCST calls.*

Proof. Because the blocks partition the nodes and the running time for each block is superlinear in the number of nodes in the block, the worst case is when there is only one block. Thus, we analyze this case. Rename the nodes so that $w_1 \leq w_2 \leq \cdots \leq w_n$. By construction of the block, $w_{i+1} \leq 8n^3 w_i$ for each i. Thus, $W \leq n w_n \leq (8n^3)^n w_{\min}$, so $O(\log \frac{W}{w_{\min}}) = O(n \log n)$. Because $\frac{c_{\max}}{c_{\min}} \leq \frac{8n^2W}{w_{\min}}$, we have $O(\log \frac{c_{\max}}{c_{\min}}) = O(n \log n)$ as well. Thus, by Corollary 9.5, each critical search takes $O(\log \frac{nWc_{\max}}{w_{\min} c_{\min}}) = O(n \log n)$ PCST calls, and there are $O(\log \frac{nc_{\max}}{c_{\min}}) = O(n \log n)$ relevant threshold values that require critical searches, for a total of $O(n^2 \log^2 n)$ PCST calls.

In analyzing the approximation ratio, we must account for the possibility of multiple blocks. First, let $OPT(B_i)$ denote the total latency of the optimal tour that starts at r and visits only the nodes in B_i. If $w'_{\min}$ is the minimum weight of any node in block B_i and $c'_{\max}$ is the distance from the root to the farthest node in B_i, then $w'_{\min} c'_{\max} \leq OPT(B_i)$. Moreover, $OPT(B_i)$ is a lower bound on the total latency of the nodes in B_i in the optimal tour that visits all nodes in the original instance. Thus, $OPT(B_1) + \cdots + OPT(B_\ell) \leq OPT$. Moreover, we will charge the latency of our overall tour block by block but in two pieces. First, we will account for the latency of block B_i's subtour, as if it were traversed on its own. Second, we will account for the amount that traversing B_i's subtour adds to the latencies of the subtours $B_{i-1}, \ldots, B_1$ that come afterward in our final tour. Both costs will be charged against $OPT(B_i)$.

The expected latency of the subtour we generated for block B_i's amended instance is at most $(2-\frac{1}{2n})\gamma OPT(B_i)$. Visiting the close deleted nodes in arbitrary order adds at most $\frac{c'_{\max} w'_{\min}}{4n} \leq \frac{OPT(B_i)}{4n}$ to the total weighted latency of all nodes in block B_i. Thus, the total latency of B_i's subtour, if it were done in isolation, would be at most $[(2-\frac{1}{2n})\gamma + \frac{1}{4n}]OPT(B_i)$.

No matter what subtour we generated for block B_i, its total length is at most $2nc'_{\max}$. The total weight of all blocks $B_{i-1}, \ldots, B_1$ is at most $\frac{w'_{\min}}{8n^2}$, so putting block B_i before them all adds at most $\frac{c'_{\max} w'_{\min}}{4n} \leq \frac{OPT(B_i)}{4n}$ to their total latency. Thus, the total amount that our subtour for block B_i adds to the weighted latency of the overall tour is at most $(2-\frac{1}{4n})\gamma OPT(B_i)$. Summing this over the blocks gives the desired result. ∎

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

Derandomizing this algorithm using the technique of section 7 with $p = O(n)$ yields the following result.

COROLLARY 8.4. *There is a deterministic algorithm for the weighted MLP that achieves an approximation factor of* 2γ *in time dominated by* $O(n^3 \log^2 n)$ *PCST calls or a factor of* $2\gamma(1+\epsilon)$ *using only* $O(\frac{1}{\epsilon} n^2 \log^2 n)$ *PCST calls.*

We note that one can obtain a deterministic $2\gamma(1+\epsilon)$-approximation for the weighted MLP using only $O(\frac{1}{\epsilon} n^2 \log^2 \frac{1}{\epsilon})$ PCST calls, by dividing up the input nodes into blocks more cleverly and then into subblocks based on their distances from the root. However, since the description and analysis of this algorithm is significantly more complex and the payoff is just to shave off a $\log^2 n$ factor in the running time, we will omit the details.

9. The critical search primitive. In this section, we show how to implement the critical search primitive that forms the basis for most of the algorithms and analyses of the previous sections. Recall our goal: Given a size target w^* or a bound target b^*, we wish to find an interval $[\lambda_{\text{lo}}, \lambda_{\text{hi}}]$ of λ values such that

1. $b_w \le OPT_w$ for $w_{\text{lo}} \le w \le w_{\text{hi}}$,
2. $c(T(\lambda)) \le \beta b_{|T(\lambda)|}$ for $\lambda \in \{\lambda_{\text{lo}}, \lambda_{\text{hi}}\}$,
3. $b_w = b_{w_{\text{lo}}} + \frac{b_{w_{\text{hi}}} - b_{w_{\text{lo}}}}{w_{\text{hi}} - w_{\text{lo}}}(w - w_{\text{lo}})$,

where $w_{\text{lo}} \le w^* \le w_{\text{hi}}$ in the case of a size target or $b_{w_{\text{lo}}} \le b^* \le b_{w_{\text{hi}}}$ in the case of a bound target.

For $w = w_{\text{lo}}, w_{\text{hi}}$, let $\tilde{b}_w$ be the lower bound $LB(\lambda)$ on OPT_w delivered by the PCST subroutine when run with $\lambda = \lambda_{\text{lo}}, \lambda_{\text{hi}}$, respectively. Set $\delta < 1$. We aim to show that, if $\lambda_{\text{hi}} - \lambda_{\text{lo}}$ is small enough, then setting $b_w = \delta \tilde{b}_w$ (for $w = w_{\text{lo}}, w_{\text{hi}}$) satisfies properties 1–3 above. In other words, scaling down the lower bounds by a factor of δ makes the interpolated lower bounds valid. Since $c(T(\lambda)) \le (2 - \frac{1}{n-1}) b_{|T(\lambda)|}$ (for $\lambda = \lambda_{\text{lo}}, \lambda_{\text{hi}}$), property 2 will hold with $\beta = \frac{1}{\delta}(2 - \frac{1}{n-1})$. For instance, $\delta = 1 - \frac{1}{4n-1}$ is sufficient to yield $\beta = 2 - \frac{1}{2n}$.

We begin by explaining where the lower bound comes from in the PCST algorithm and why it extends to give lower bounds on OPT_w for all values of w. We model the w-MST problem as the following integer program:

$$\text{Min} \qquad \sum_{e \in E} c_e x_e$$

subject to:

$$\sum_{e \in \delta(S)} x_e + \sum_{T : T \supseteq S} z_T \ge 1 \quad \forall S \subseteq V \setminus \{r\},$$

$$\sum_{S : S \subseteq V \setminus \{r\}} w(S) z_S \le W - w,$$

$$x_e \in \{0,1\} \qquad \forall e \in E,$$

$$z_S \in \{0,1\} \qquad \forall S \subseteq V \setminus \{r\},$$

where $\delta(S)$ is the set of edges with exactly one end point in S and $w(S) = \sum_{v \in S} w_v$. The variable $x_e = 1$ indicates that the edge e is in the tree, while $z_S = 1$ indicates that the set of nodes S is not spanned. The first set of constraints says that, for any set S of nodes not containing the root, either they are contained in the unspanned set or some edge in $\delta(S)$ is selected. The second constraint says that the total weight of unspanned nodes is at most $W - w$.

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

Following [13], we can convert this to something close to a PCST problem by applying Lagrangian relaxation to the second constraint:

$$\text{Min} \quad \sum_{e\in E} c_e x_e + \lambda \left(\sum_{S:S\subseteq V\setminus\{r\}} w(S) z_S - (W-w) \right)$$

subject to:

$$\begin{aligned} \sum_{e\in\delta(S)} x_e + \sum_{T:T\supseteq S} z_T &\geq 1 && \forall S \subseteq V\setminus\{r\}, \\ x_e &\in \{0,1\} && \forall e \in E, \\ z_S &\in \{0,1\} && \forall S \subseteq V\setminus\{r\}. \end{aligned}$$

Note that any solution feasible for the previous integer program will be feasible for this one, and if $\lambda \geq 0$, it will have no greater cost in the new integer program. Recall the definition of the PCST problem: We are given an undirected graph (V, E), a root node $r \in V$, nonnegative costs on the edges $c_e \geq 0$ for all $e \in E$, and nonnegative penalties p_i for $i \in V, i \neq r$. The goal is to find a tree spanning the root node so as to minimize the cost of the edges in the tree plus the penalties of the nodes not in the tree. Here we set all penalties $p_i = \lambda w_i$. Observe that the integer program above exactly models this problem for $p_i = \lambda w_i$, except that the objective function has an additional constant term of $-(W-w)\lambda$.

Goemans and Williamson [21] give a primal-dual 2-approximation algorithm for the PCST problem. Their algorithm returns a tree spanning the root node and a solution to the dual of a linear programming relaxation of the PCST problem. The dual solution is feasible for the dual of the linear programming relaxation of the integer program above; in particular, this dual is

$$\text{Max} \quad \sum_{S\subseteq V\setminus\{r\}} y_S - \lambda(W-w)$$

subject to:

$$\text{(D)} \qquad \begin{aligned} \sum_{S:e\in\delta(S)} y_S &\leq c_e && \forall e \in E, \\ \sum_{T:T\subseteq S} y_T &\leq w(S)\lambda && \forall S \subseteq V\setminus\{r\}, \\ y_S &\geq 0 && \forall S \subseteq V\setminus\{r\}. \end{aligned}$$

We will abbreviate their algorithm as PCST. In particular, they show the following.

THEOREM 9.1 (see [21]). *PCST returns a tree T and a dual solution y feasible for* (D) *such that if X is the set of nodes not spanned by T, then*

$$\sum_{e\in T} c_e + \left(2 - \frac{1}{n-1}\right) \lambda w(X) \leq \left(2 - \frac{1}{n-1}\right) \sum_{S\subseteq V\setminus\{r\}} y_S.$$

From this theorem we obtain the following lemma.

LEMMA 9.2. *Letting T and y be the tree and dual solution returned by PCST, respectively, define*

$$\tilde{b}_w := \sum_{S\subseteq V\setminus\{r\}} y_S - \lambda(W-w).$$

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

Then $\tilde{b}_w \le OPT_w$ for each $w = 1, \ldots, W$, and the cost of T is no more than $(2 - \frac{1}{n-1})\tilde{b}_{w(T)}$, where $w(T)$ is the total weight of the nodes spanned by T.

Proof. Note that if y is a feasible dual solution to (D), then since $\sum_{S \subseteq V \setminus \{r\}} y_S - \lambda(W - w)$ is the dual objective function of (D), it is a lower bound on the cost of an optimal w-MST. By Theorem 9.1, if PCST returns tree T and X is the set of nodes not spanned by T, then $w(X) = W - w(T)$. Thus

$$\sum_{e \in T} c_e + \left(2 - \frac{1}{n-1}\right) \lambda(W - w(T)) \le \left(2 - \frac{1}{n-1}\right) \sum_{S \subseteq V \setminus \{r\}} y_S,$$

which implies that

$$\sum_{e \in T} c_e \le \left(2 - \frac{1}{n-1}\right) \left(\sum_{S \subseteq V \setminus \{r\}} y_S - \lambda(W - w(T)) \right) = \left(2 - \frac{1}{n-1}\right) \tilde{b}_{w(T)}. \qquad \square$$

Notice that running the PCST algorithm for a single value of λ yields lower bounds on OPT_w for *every* value of w simultaneously.

We further need the following observation, which relies on the workings of the PCST algorithm.

Observation 9.3. If we call PCST with $\lambda = 0$, it will return a tree containing only the root node. If we call PCST with $\lambda = \frac{c_{\max}}{w_{\min}}$, where $c_{\max}$ is the maximum edge cost and $w_{\min}$ is the minimum node weight, it will return a tree spanning all nodes.

We are now ready to prove our crucial lemma.

LEMMA 9.4. *Let T_{lo}, T_{hi} be the trees and $\tilde{b}_{\text{lo}}, \tilde{b}_{\text{hi}}$ the lower bounds returned by PCST when the penalty parameter λ is set to λ_{lo} and λ_{hi}, respectively, with $\lambda_{\text{lo}} \le \lambda_{\text{hi}}$ and $\lambda_{\text{hi}} - \lambda_{\text{lo}} \le \frac{1-\delta}{\delta} \frac{c_{\min}}{W}$. Let $w \in (w(T_{\text{lo}}), w(T_{\text{hi}}))$, and express w as a convex combination $w = \alpha_{\text{lo}} w(T_{\text{lo}}) + \alpha_{\text{hi}} w(T_{\text{hi}})$, where $\alpha_{\text{lo}} + \alpha_{\text{hi}} = 1$. If we set $b_{\text{lo}} = \delta \tilde{b}_{\text{lo}}, b_{\text{hi}} = \delta \tilde{b}_{\text{hi}}$, and $b_w = \alpha_{\text{lo}} b_{\text{lo}} + \alpha_{\text{hi}} b_{\text{hi}}$, then $b_w \le OPT_w$.*

Proof. Let $w_{\text{lo}} = w(T_{\text{lo}})$ and $w_{\text{hi}} = w(T_{\text{hi}})$. Let y^{lo} and y^{hi} be the dual solutions returned by PCST for penalty value λ_{lo} and λ_{hi}, respectively. Letting $y = \alpha_{\text{lo}} y^{\text{lo}} + \alpha_{\text{hi}} y^{\text{hi}}$, observe that (y, λ_{hi}) is feasible for (D) by the convexity of the feasible region, and thus both $\sum_{S \subseteq V \setminus \{r\}} y_S - (W - w)\lambda_{\text{hi}}$ and $c_{\min}$ are lower bounds on the cost of an optimal w-MST. Then

$$\begin{aligned}
b_w &= \alpha_{\text{lo}} b_{\text{lo}} + \alpha_{\text{hi}} b_{\text{hi}} \\
&= \delta \left(\alpha_{\text{lo}} \tilde{b}_{\text{lo}} + \alpha_{\text{hi}} \tilde{b}_{\text{hi}} \right) \\
&= \delta \left(\alpha_{\text{lo}} \left(\sum_{S \subseteq V \setminus \{r\}} y_S^{\text{lo}} - (W - w_{\text{lo}})\lambda_{\text{lo}} \right) + \alpha_{\text{hi}} \left(\sum_{S \subseteq V \setminus \{r\}} y_S^{\text{hi}} - (W - w_{\text{hi}})\lambda_{\text{hi}} \right) \right) \\
&= \delta \left(\sum_{S \subseteq V \setminus \{r\}} y_S - \alpha_{\text{lo}}(W - w_{\text{lo}})(\lambda_{\text{hi}} + \lambda_{\text{lo}} - \lambda_{\text{hi}}) - \alpha_{\text{hi}}(W - w_{\text{hi}})\lambda_{\text{hi}} \right) \\
&\le \delta \left(\sum_{S \subseteq V \setminus \{r\}} y_S - (W - w)\lambda_{\text{hi}} + \alpha_{\text{lo}}(W - w_{\text{lo}}) \frac{1 - \delta}{\delta} \frac{c_{\min}}{W} \right) \\
&\le \delta \left(OPT_w + \frac{1 - \delta}{\delta} OPT_w \right) \\
&\le OPT_w. \qquad \square
\end{aligned}$$

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

COROLLARY 9.5. *Given $\delta < 1$ and $\beta \geq \frac{1}{\delta}(2 - \frac{1}{n-1})$, a single critical search can be performed using $O(\log \frac{\delta c_{\max} W}{(1-\delta) c_{\min} w_{\min}})$ PCST calls.*

Proof. Start the binary search with the interval $[0, \frac{c_{\max}}{w_{\min}}]$. Each PCST call halves the interval, and by Lemma 9.4, we can stop once we shrink the interval to width $\frac{(1-\delta)c_{\min}}{\delta W}$. □

In particular, if we set $\delta = 1 - \frac{1}{4n-1}$ in order to achieve $\beta = 2 - \frac{1}{2n}$, then each critical search takes $O(\log \frac{n c_{\max} W}{c_{\min} w_{\min}})$ PCST calls.

10. Experimental results. The approximation ratio of our algorithm for real instances is much better then one could expect by the worst-case analysis. We tested our deterministic algorithm of section 6 on some 2-dimensional Euclidean instances available from the TSPLIB [33]. In Table 2 we show the experimental factor between the obtained tour latency and the lower bound. The average ratio on the tested instances is 3.01, and its maximum is 3.66. We note that these numbers are much less than $2\gamma \approx 7.18$, which is the theoretical worst case.

11. Concluding remarks. We showed how to use the tree concatenation technique of Blum et al. as refined by Goemans and Kleinberg to construct a $2\gamma \approx 7.18$-approximation algorithm for the MLP, while having access to 2-approximate k-MSTs for only a few values of k that we cannot specify in advance. The 2γ guarantee comes from two sources. The 2 comes from our k-MSTs being 2-approximate, while the γ comes from the tree concatenation procedure. Both of these pieces represent significant barriers to further improvement.

The factor of $\gamma \approx 3.59$ from the tree concatenation is inherent in any analysis that blindly concatenates trees and upper bounds the latency by the sum of modified latencies. This is because Goemans and Kleinberg prove that Theorem 4.3 is tight; that is, the costs $d_2, \ldots, d_n$ can be selected such that the ratio of shortest path length in the graph G to $d_2 + \cdots + d_n$ is arbitrarily close to γ. Thus, in order to attain a provably better latency, one would need to either have some knowledge of the costs d_i or pay attention to the actual structure of the trees being concatenated.

All known constant factor approximation algorithms for the k-MST problem rely explicitly or implicitly on the LP relaxation we used for the PCST problem. Since this relaxation has an integrality gap of essentially 2, it seems that achieving a β-approximation algorithm for k-MST for a constant $\beta < 2$ will require a significantly different approach.

As we mentioned in section 1, Chaudhuri et al. [12] get around this difficulty by using the optimal k-stroll rather than the optimal k-MST as their lower bound on the latency of the kth node visited in the tour. Our work paved the way for theirs, since they use our technique of bluffing the GK algorithm with phantom trees, in order to make up for the fact that the Lagrangian relaxation will typically fail to yield trees of some sizes. Their algorithm requires them to guess the end point of the k-stroll, which incurs an extra factor of n in their running time. Their improvement comes because this allows them to raise the coefficient of nearly all of their dual variables from 1 to 2 in the objective function of the dual LP, which is a slight variant of (D). They use the analogous variant of the PCST algorithm to find their tree and dual solution. This is essentially the same trick used by Goemans and Williamson in [21] to transform their PCST algorithm into a 2-approximation for the prize-collecting TSP.

One direction for future work would be to consider LP relaxations that address the MLP objective function directly rather than using k-MSTs or k-strolls for our lower bounds. Perhaps the most attractive special case to look at is where the underlying

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

TABLE 2
Experimental results of our deterministic algorithm from section 6. Running times are in CPU seconds on an IBM RS/6000 43P Model 140.

Instance name	Tour latency	Lower bound	Factor	Running time
berlin52	197137	58644	3.36	0.983
bier127	5929120	1886700	3.14	4.033
ch130	455849	148344	3.07	11.033
ch150	571369	209537	2.72	13.833
d198	1380470	556278	2.48	21.983
d493	10441397	3305791	3.15	144.150
d657	20831492	6487270	3.21	288.000
eil101	38582	12157	3.17	2.900
eil51	14683	4390	3.34	0.383
eil76	26128	8046	3.24	1.017
fl417	2531146	825513	3.06	258.867
gil262	393641	126697	3.10	34.183
kroA100	1307340	432542	3.02	3.683
kroA150	2494782	811515	3.07	8.967
kroA200	3387616	1173404	2.88	40.667
kroB100	1274207	442308	2.88	2.083
kroB150	2376125	820770	2.89	6.883
kroB200	3731218	1174833	3.17	44.083
kroC100	1207746	432224	2.79	4.700
kroD100	1297932	412501	3.14	4.450
kroE100	1345314	446334	3.01	1.800
lin105	780662	274250	2.84	4.767
lin318	7475822	2532401	2.95	128.750
p654	10251922	3545177	2.89	408.000
pcb442	14683399	4844532	3.03	79.133
pr1002	164844296	50583204	3.25	1479.983
pr107	2205490	915582	2.40	124.350
pr124	4778217	1454570	3.28	7.817
pr136	8720053	2891809	3.01	30.967
pr144	4844537	1674418	2.89	30.700
pr152	6075505	2334659	2.60	49.333
pr226	10421449	3283953	3.17	15.750
pr264	7674241	2628452	2.91	26.517
pr299	8553790	2938150	2.91	109.250
pr439	24126010	7900826	3.05	143.083
pr76	4359810	1467212	2.97	1.350
rat195	280900	102741	2.73	25.850
rat575	2511713	847350	2.96	418.833
rat783	4410164	1527124	2.88	449.983
rat99	75048	25964	2.89	5.033
rd100	458419	153887	2.97	5.250
rd400	3930767	1230238	3.19	93.517
st70	26384	9033	2.92	0.800
ts225	17953213	6271875	2.86	2815.500
tsp225	537080	181263	2.96	88.733
u1060	146511585	46213643	3.17	753.283
u159	3837650	1301475	2.94	9.433
u574	12906940	4159616	3.10	195.233
u724	19821239	6222958	3.18	517.433
vm1084	153128900	41816544	3.66	1041.333

metric is given by a tree. Since the k-MST problem can be solved optimally on trees, Goemans and Kleinberg [22] used their algorithm to obtain a 3.59-approximation for this special case, without resorting to the k-stroll approach of [12]. This is still the best result known for tree metrics.

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

Acknowledgments. A preliminary version of this paper by the first and third authors [4] had a performance guarantee of 9.28. The second author contributed the core idea for the improvement of the performance guarantee to 7.18, which first appeared in [3]. We thank Tim Roughgarden for many enlightening discussions and Martin Pál, David Shmoys, and Éva Tardos for helpful comments on our presentation.

REFERENCES

[1] F. Afrati, S. Cosmadakis, C. H. Papadimitriou, G. Papageorgiou, and N. Papakostantinou, *The complexity of the traveling repairman problem*, Informatique Theorique et Applications, 20 (1986), pp. 79–87.

[2] S. R. Agnihothri, *A mean value analysis of the travelling repairman problem*, IIE Transactions, 20 (1988), pp. 223–229.

[3] A. Archer, A. Levin, and D. P. Williamson, *A faster, better approximation algorithm for the minimum latency problem*, Technical report 1362, Cornell University ORIE, 2003.

[4] A. Archer and D. P. Williamson, *Faster approximation algorithms for the minimum latency problem*, in Proceedings of the 14th Annual ACM-SIAM Symposium on Discrete Algorithms, 2003, pp. 88–96.

[5] S. Arora and G. Karakostas, *Approximation schemes for minimum latency problems*, SIAM J. Comput., 32 (2003), pp. 1317–1337.

[6] S. Arora and G. Karakostas, *A* $2+\epsilon$ *approximation algorithm for the k-MST problem*, Math. Program., 107 (2006), pp. 491–504.

[7] S. Arya and H. Ramesh, *A* 2.5 *factor approximation algorithm for the k-MST problem*, Inform. Process. Lett., 65 (1998), pp. 117–118.

[8] G. Ausiello, S. Leonardi, and A. Marchetti-Spaccamela, *On salesmen, repairmen, spiders and other traveling agents*, in Proceedings of the Italian Conference on Algorithms and Complexity, Lecture Notes in Comput. Sci., 1767, Springer, Berlin, 2000, pp. 1–16.

[9] I. Averbakh and O. Berman, *Sales-delivery man problems on treelike networks*, Networks, 25 (1995), pp. 45–58.

[10] L. Bianco, A. Mingozzi, and S. Ricciardelli, *The traveling salesman problem with cumulative costs*, Networks, 23 (1993), pp. 81–91.

[11] 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, 1994, pp. 163–171.

[12] K. Chaudhuri, B. Godfrey, S. Rao, and K. Talwar, *Paths, trees, and minimum latency tours*, in Proceedings of the 44th Annual IEEE Symposium on the Foundations of Computer Science, 2003, pp. 36–45.

[13] F. A. Chudak, T. Roughgarden, and D. P. Williamson, *Approximate k-MSTs and k-Steiner trees via the primal-dual method and Lagrangean relaxation*, Math. Program., 100 (2004), pp. 411–421.

[14] J. Fakcharoenphol, C. Harrelson, and S. Rao, *The k-traveling repairman problem*, in Proceedings of the 14th Annual ACM-SIAM Symposium on Discrete Algorithms, 2003, pp. 655–664.

[15] E. Feuerstein and L. Stougie, *On-line single-server dial-a-ride problems*, Theoret. Comput. Sci., 268 (2001), pp. 91–105.

[16] M. Fischetti, G. Laporte, and S. Martello, *The delivery man problem and cumulative matroids*, Oper. Res., 41 (1993), pp. 1065–1064.

[17] F. V. Fomin and A. Lingas, *Approximation algorithms for time-dependent orienteering*, Inform. Process. Lett., 83 (2002), pp. 57–62.

[18] H. N. Gabow and S. Pettie, *The dynamic vertex minimum problem and its application to clustering-type approximation algorithms*, in Proceedings of the 8th Scandinavian Workshop on Algorithm Theory, Lecture Notes in Comput. Sci. 2368, Springer, Berlin, 2002, pp. 190–199.

[19] N. Garg, *A* 3*-approximation for the minimum tree spanning k vertices*, in Proceedings of the 37th Annual Symposium on Foundations of Computer Science, IEEE, Piscataway, NJ, 1996, pp. 302–309.

[20] N. Garg, *Saving an epsilon: A* 2*-approximation for the k-MST problem in graphs*, in Proceedings of the 37th Annual ACM Symposium on Theory of Computing, 2005, pp. 396–402.

[21] M. X. Goemans and D. P. Williamson, *A general approximation technique for constrained forest problems*, SIAM J. Comput., 24 (1995), pp. 296–317.

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

[22] M. Goemans and J. Kleinberg, *An improved approximation ratio for the minimum latency problem*, Math. Program., 82 (1998), pp. 111–124.
[23] R. Hassin and A. Levin, *An approximation algorithm for the minimum latency set cover problem*, in Proceedings of the 13th Annual European Symposium on Algorithms, Lecture Notes in Comput. Sci. 3669, Springer, Berlin, 2005, pp. 726–733.
[24] K. Jain and V. V. Vazirani, *Approximation algorithms for metric facility location and k-median problems using the primal-dual schema and Lagrangian relaxation*, J. ACM, 48 (2001), pp. 274–296.
[25] E. Koutsoupias, C. H. Papadimitriou, and M. Yannakakis, *Searching a fixed graph*, in Proceedings of the 23rd International Colloquium on Automata, Languages, and Programming, Lecture Notes in Comput. Sci. 1099, Springer, Berlin, 1996, pp. 280–289.
[26] S. O. Krumke, W. de Paepe, D. Poensgen, and L. Stougie, *News from the online traveling repairman*, Theoret. Comput. Sci., 295 (2003), pp. 279–294.
[27] G. Lin, C. Nagarajan, R. Rajaraman, and D. P. Williamson, *A general approach for incremental approximation and hierarchical clustering*, in Proceedings of the 17th Annual ACM-SIAM Symposium on Discrete Algorithms, 2006, pp. 1147–1156.
[28] A. Lucena, *Time-dependent traveling salesman problem - the deliveryman case*, Networks, 20 (1990), pp. 753–763.
[29] N. Megiddo, *Combinatorial optimization with rational objective functions*, Math. Oper. Res., 4 (1979), pp. 414–424.
[30] E. Minieka, *The delivery man problem on a tree network*, Ann. Oper. Res., 18 (1989), pp. 261–266.
[31] C. H. Papadimitriou and M. Yannakakis, *The traveling salesman problem with distances one and two*, Math. Oper. Res., 18 (1993), pp. 1–11.
[32] J.-C. Picard and M. Queyranne, *The time-dependent traveling salesman problem and its application to the tardiness problem in one-machine scheduling*, Oper. Res., 26 (1978), pp. 86–110.
[33] G. Reinelt, *TSPLIB*, Technical report, Universität Heidelberg Institut für Informatik, Im Neuenheimer Feld 368, D-69120 Heidelberg, Germany.
[34] S. Sahni and T. Gonzalez, *P-complete approximation problems*, J. ACM, 23 (1976), pp. 555–565.
[35] D. Simchi-Levi and O. Berman, *Minimizing the total flow time of n jobs on a network*, IIE Transactions, 23 (1991), pp. 236–244.
[36] R. Sitters, *The minimum latency problem is NP-hard for weighted trees*, in Proceedings of the 9th Conference on Integer Programming and Combinatorial Optimization, Lecture Notes in Comput. Sci. 2337, Springer, Berlin, 2002, pp. 230–239.
[37] J. N. Tsitsiklis, *Special cases of traveling salesman and repairman problems with time windows*, Networks, 22 (1992), pp. 263–282.
[38] R. J. Vander Wiel and N. V. Sahinidis, *Heuristic bounds and test problem generation for the time-dependent traveling salesman problem*, Transportation Science, 29 (1995), pp. 167–183.
[39] R. J. Vander Wiel and N. V. Sahinidis, *An exact solution approach for the time-dependent traveling-salesman problem*, Naval Res. Logist., 43 (1996), pp. 797–820.
[40] I. R. Webb, *Depth-first solutions for the deliveryman problem on tree-like networks: An evaluation using a permutation model*, Transportation Science, 30 (1996), pp. 134–147.
[41] B. Wu, *Polynomial time algorithms for some minimum latency problems*, Inform. Process. Lett., 75 (2000), pp. 225–229.
[42] C. Yang, *A dynamic programming algorithm for the travelling repairman problem*, Asia-Pac. J. Oper. Res., 6 (1989), pp. 192–206.

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.