Paths, Trees, and Minimum Latency Tours

Kamalika Chaudhuri* Brighten Godfrey† Satish Rao‡ Kunal Talwar§

Abstract

We give improved approximation algorithms for a variety of latency minimization problems. In particular, we give a 3.59[1]*-approximation to the minimum latency problem, improving on previous algorithms by a multiplicative factor of 2. Our techniques also give similar improvements for related problems like k-traveling repairmen and its multiple depot variant. We also observe that standard techniques can be used to speed up the previous and this algorithm by a factor of $\tilde{O}(n)$.*

1 Introduction

We study the *minimum latency problem* (MLP), which is the problem of finding a tour of a set of points in a metric space which minimizes the sum of the latencies to the points, where the latency of a point is its distance along the tour. This problem is **NP**-complete even when the metric space is induced by a tree [30], and is **MaxSNP**-hard in general graphs. It has also been referred to in literature as the *traveling repairman problem* [1], the *school-bus driver problem* [34], and the *delivery man problem* [17, 26].

For general metrics, with which we are concerned, Blum et.al. [9] gave the first constant factor approximation. This was improved by Goemans and Kleinberg [21]. Both algorithms proceed by finding tours of geometrically increasing costs and stitching them together. Moreover, these algorithms both use certain approximable lower bounds that were subsequently replaced with the k-MST problem: the problem of finding a minimum cost tree containing k nodes. The cost of a k-MST is a lower bound on the cost of any tour that covers k nodes, and thus on the latency of the kth node in the minimum latency tour.

Approximations for the k-MST problem also evolved with polylogarithmic approximations and include the constant factor approximation given by Blum, Ravi and Vempala [11] and the 3-approximation of Garg [20]. Currently, one can find essentially 2-approximate solutions [5, 3] for the k-MST. The combination of these ideas gives a 7.18-approximation algorithm for the minimum latency problem, which arises from a factor of 3.59 from techniques in [21] and the factor of two from the k-MST algorithms.

Again, the MLP approximation algorithms lower bound the latency of the k^{th} vertex in the tour by the cost of the minimum tree spanning k vertices, that is, the k-MST. We note that the cost of the minimum path visiting k vertices is a better lower bound on the latency of the k^{th} vertex in the tour. However, since the problem of finding the cheapest such path (we call it the k-stroll problem) is not known to be better approximable than the k-MST, we cannot hope to improve the approximation guarantee by finding an approximate k-stroll. The basic idea then is to find a good tree spanning k vertices, and bound its cost in comparison to the optimum k-stroll. We give a primal-dual algorithm based on [20, 5] that guarantees we find a k-tree whose cost is no more than the optimal k-stroll. In particular, our primal-dual algorithm outputs a primal integral solution to the k-MST problem, and a feasible dual solution (of no smaller cost) to the k-stroll problem.

Thus, we do not pay the factor of two for the k-MST algorithms, since our tree has no more cost than the k-stroll. We are left with an approximation factor of 3.59 that arises from the techniques of Goemans and Kleinberg [21]. Since algorithms for various related problems such as k-traveling repairman [14, 12] and Orienteering and discounted reward TSP [10] use similar lower bounds, we get improved approximation algorithms for these problems as well.

We note that, in addition to producing larger lower

*Computer Science Division, University of California, Berkeley. Email: kamalika@cs.berkeley.edu.

†Computer Science Division, University of California, Berkeley. Email: pbg@cs.berkeley.edu.

‡Computer Science Division, University of California, Berkeley. Email:satishr@cs.berkeley.edu. Research partially supported by the NSF via grant CCR-0105533.

§Computer Science Division, University of California, Berkeley. Email: kunal@cs.berkeley.edu. Research partially supported by the NSF via grants CCR-0121555 and CCR-0105533.

[1]More precisely, this factor is γ given by the root of the equation $\frac{\gamma+1}{\ln\gamma} = \gamma$. In the rest of the paper, we shall use 3.59 instead of γ.

bounds on the optimal tour, our algorithm may produce better k-trees. In particular, our method for finding a k-tree guesses the endpoints of a path and forces them to be in the k-tree. That is, we produce k-trees that are better than those produced by the currently known approximate k-MST algorithm used by Archer, Levin, and Williamson [3]. The algorithm of Arora and Karakostas [5], however, also guesses points to include for a different reason: in order to efficiently interpolate to a tree containing exactly k nodes (which, in fact, is not necessary for MLP). Our guessing is required for finding trees that consist of a far away dense cluster of nodes, which traditional primal-dual k-MST approximation algorithms completely miss.

Finally, we note that good lower bounds significantly affect running times for branch and bound techniques. While a factor of two may be a significant improvement as a lower bound, we suspect that our reliance on the piecing together of tours remains a serious obstacle to having a truly useful lower bounding technique for branch and bound.

1.1 Related work

Sahni and Gonzales [28] showed that the minimum latency problem is **NP**-hard. It was shown to be **MaxSNP**-hard by a reduction from the Traveling Salesman Problem with distances 1 and 2 [27, 9]. Indeed, the problem remains **NP**-hard even on weighted trees as shown by Sitters [30].

For the unweighted case (i.e. for a shortest path metric on an unweighted graph), Koutsoupias, Papadimitriou and Yannakakis [23] give a 1.662-approximation.

As mentioned above, for general metrics, the algorithm of Goemans and Kleinberg could use the k-MST algorithm of Arora and Karakostas [5], a $(2+\epsilon)$-approximation in time $O(n^{1/\epsilon})$. The algorithm of [5] builds on the algorithm of Garg [20], which shows how to interpolate solutions to the prize collecting Steiner tree (PCST) problem (which is the Lagrangian relaxation of the k-MST problem). Archer, Levin and Williamson [3] improve the bound a bit and the running time substantially to[2] $\tilde{O}(n^3)$ by showing that the 2-approximate solutions returned by the PCST subroutine actually suffice for the minimum latency algorithm. They use $n \log n$ instances of the PCST problem (see [22]). The PCST has a 2-approximation algorithm and its running time was recently improved to $O(n^2)$ in [19].

For weighted trees and points in $\mathbb{R}^d$, Arora and Karakostas [4] gave a quasipolynomial time approximation scheme. We note that this result actually avoids the limitation of stitching together tours.

A generalization of the minimum latency problem where k tours cover (or k repairmen visit) the points was studied by Fakcharoenphol, Harrelson and Rao [14]. They call it the k-traveling repairman problem. The latency of a point is the distance from the starting point along the tour that covers it. They give a 16.994-approximation to this problem. Chekuri and Kumar [12] give a 24-approximation for a “multiple depot” version of this problem where the repairman can start at differing locations.

In the operations research community, there are several exact exponential time algorithms for the minimum latency problem, e.g. [36, 17, 29, 8, 25]. Researchers have also evaluated various heuristic approaches [33, 32] and studied stochastic [2] and online versions [16, 24]. Finally, there has been work on various special cases [1, 26, 7, 31, 35].

In addition to the self-evident applications, researchers have applied the minimum latency problem to minimizing the time to search for a treasure on a fixed graph (such as the searching the web graph); see e.g. Koutsoupias et.al. [23] and Ausiello et.al. [6].

Finally, we note that the minimum latency problem is related to the minimum sum set cover problem where one wishes to find the set cover which minimizes the total latency of covering the underlying elements. Feige, Lovász, and Tetali [15] gave a 4-approximation for this problem and showed that one cannot do better unless **NP** is contained in quasipolynomial time. Thus, our result shows that the minimum latency problem is easier than the minimum sum set cover problem.

1.2 Our results

As described above, we produce a 3.59-approximation algorithm for the minimum latency problem.

Our algorithm proceeds by repeatedly using a primal-dual algorithm that finds a k-tree containing two specified points. As in previous algorithms, our primal-dual algorithm actually solves a Lagrangian relaxation of the original problem. Using the techniques of Garg [20], Arora and Karakostas [5], we can solve the original problem with approximation guarantee $(1+\epsilon)$. Better still, we can argue, as in Archer, Levin and Williamson [3] that just using the solutions to the Lagrangian relaxation(s) actually suffices.

We also observe that the running time of the Archer, Levin and Williamson [3] algorithm can be improved to $\tilde{O}(n^2)$ due to the fact that the Goemans-Kleinberg analysis only uses $O(\log n)$ trees. Our running times are slightly worse: for the minimum latency problem,

[2] We shall suppress polylogarithmic factors in the $\tilde{O}$ notation.

we obtain in $\tilde{O}(n^3)$ time $(3.59+\epsilon)$-approximation algorithm and an $\tilde{O}(n^4)$ time 3.59-approximation algorithm.

The $(2+\epsilon)$-approximation algorithm for the k-stroll problem that we describe in Section 4 is a useful subroutine for some other applications as well. For the k-traveling repairman problem, the approximation factor given by Fakcharoenphol, Harrelson and Rao [14] improves by a factor of two to 8.49. For the multiple depot version of k-traveling repairman, the approximation factor of 24 given by Chekuri and Kumar [12] improves to 12. Moreover, our algorithm can give stronger guarantees on the path length, in terms of "excess" (defined as length of the path *minus* distance from source to destination). This improves the results of Blum et.al. [10] on orienteering and discounted reward TSP.

The rest of the paper is organized as follows. In Section 2, we give definitions and a short overview of our algorithm. In Section 3, we show that if, for every k, we could find a k-tree of cost no more than the best k-stroll, we would get a 3.59-approximation for the minimum latency problem. This argument closely follows that used by Goemans and Kleinberg [21] for their 3.59-approximation for minimum latency on trees (and $7.18+\epsilon$ on arbitrary metrics). In Section 4, we show how to obtain such k-trees for some values of k. This is the main technical section, and uses a subtle variation of the classical primal-dual argument. In Section 5, we show, along the lines of Archer, Levin and Williamson [3], that it actually suffices to use the few k-trees that we found using the primal-dual algorithm.

2 Preliminaries

2.1 Definitions

Let $G = (V, E, c)$ be a weighted undirected graph, and let $P = (v_1, \ldots, v_n)$ be a path (or tour) in that graph. We use $c(P)$ to mean the total weight of all edges in P. The *latency* $l_{v_i,P}$ of a vertex $v_i \in P$ along path P is the cost of the prefix of P ending at v_i; that is, $l_{v_i,P} = c((v_1, \ldots, v_i))$. Usually the path in question will be implicit, so we simply write l_{v_i}.

We define the following in terms of G and a source node $s \in V$ which will often be implicit. Our main problem is the *minimum latency problem* (MLP): Find a tour of G beginning at s which minimizes the sum of the latencies of the nodes. A *k-tree* (usually denoted T_k), *k-tour*, or *k-stroll* is a tree, tour, or path, respectively, containing at least k vertices and beginning at s. A *k-path from s to t* is a path from s to t containing at least k vertices. Note that the optimum (minimum cost) k-stroll in any graph is the minimum over all t of the optimum k-path from s to t.

k-MST is the problem of finding a minimim cost k-tree. Unless otherwise stated, we deal with the rooted version of the problem which must include s.

2.2 Algorithm outline

Our algorithm for the minimum latency problem proceeeds as follows. For every t and for a suitable set of values of k, we run the algorithm of Section 4 to get "low cost" trees containing s and t and spanning at least k vertices. We then convert these trees to tours by traversing them in some order, to form our pool of tours. We then construct an auxiliary graph with these tours as vertices and find the shortest path between two fixed nodes (Section 3). Finally, we traverse the tours corresponding to vertices on the shortest path to get our final tour.

3 From good k-trees to a low latency tour

Call a k-tree *good* if it costs no more than the optimal k-stroll. In this section, we show that if we could find a good k-tree for every k, then we could find a 3.59-approximate solution to the minimum latency problem. Note that a k-tree can be traversed to get a k-tour of cost no more than twice that of the tree.

Call a k-tour *good* if it costs no more than twice as much as the optimal k-stroll. We assume that we are given good k-tours for every k. Our approximate minimum latency tour is built up by stitching together a suitable combination of good tours.

We present an algorithm SP [3] which finds such a suitable combination. The input to the algorithm is a set of tours T_j for $j = 1, 2, \ldots, n$, where T_j is a good j-tour. We build a weighted graph H whose vertices are the integers $1, \ldots, n$, where vertex i corresponds to tour T_i. The edges of H are $i \to j$ for $j > i$. The cost of edge $i \to j$ reflects how much it would cost to follow tour T_j after tour T_i. With some foresight, we set this cost to be $(n - \frac{i+j}{2})c(T_j)$. The approximate minimum latency tour returned by SP is comprised of the tours along the shortest path from 1 to n in H. We choose a random traversal direction for each tour along the shortest path, and concatenate these tours to get our final tour T. The following claim is due to Goemans and Kleinberg [21].

Claim 1 *Suppose T is composed of the subtours $T_{n_1}, \ldots, T_{n_t}$. Then the expected sum of the latencies of the vertices along T is at most the length of the path $n_1 \to n_2 \to \ldots \to n_t$ in H.*

Proof: Consider the k^{th} vertex in the concatenated tour, where $n_{i-1} < k \leq n_i$. This vertex will occur in or

[3] SP here stands for Shortest Path.

before tour T_{n_i}. Its expected latency in the tour can be therefore upper bounded by

$$\mathrm{E}[l_{v_k}] \quad \le \quad \frac{1}{2}c(T_{n_i}) + \sum_{l=1}^{i-1} c(T_{n_l}).$$

Let us call this upper bound u_k. The expected sum of the latencies of all the vertices in the tour is at most

$$\begin{aligned}
&\sum_{k=1}^{n} \mathrm{E}[l_{v_k}] \\
&\quad \le \quad \sum_{k=1}^{n} u_k \\
&\quad = \quad \sum_{i=1}^{t} \sum_{n_{i-1}<k\le n_i} u_k \\
&\quad = \quad \sum_{i=1}^{t} (n_i - n_{i-1}) \left(\frac{1}{2}c(T_{n_i}) + \sum_{l=1}^{i-1} c(T_{n_l}) \right) \\
&\quad = \quad \sum_{i=1}^{t} \frac{1}{2}(n_i - n_{i-1})c(T_{n_i}) + (n - n_i)c(T_{n_i}) \\
&\quad = \quad \sum_{i=1}^{t} \left(n - \frac{n_i + n_{i-1}}{2} \right) c(T_{n_i})
\end{aligned}$$

which is the cost of the path $n_1 \to n_2 \to \ldots \to n_t$ in H. In the worst case, the tours are nested, and for $n_{i-1} < k \le n_i$, the k^{th} vertex appears in the i^{th} tour. The equalities in the fourth and fifth lines follow from a little reorganisation of the terms. ■

We now claim that the length of the shortest path in H is at most 3.59 times the sum of the latencies along the optimal tour. We will justify our claim by showing a path in H that costs as much.

Let T be a tour formed by concatenating the sequence of subtours $T_{n_1}, T_{n_2}, \ldots$ in which the i^{th} subtour T_{n_i} is a good tour of length at most $2bc^i$ containing the maximum number of vertices. Here c is a parameter greater than 1, the exact value of which is to be chosen later. We set $b = c^U$, where U is a random variable distributed uniformly between 0 and 1. From the proof of Claim 1, it follows that the cost of the corresponding path in H is the sum of the upper bounds u_i on the latencies of the vertices.

We will bound this upper bound u_i on the latency of the i^{th} vertex v_i in T in terms of the latency of the i^{th} vertex v_i^{opt} in the optimal tour. Suppose $l_{v_i^{opt}} = dc^j$, where $d < c$. There can be two cases: $d < b$ and $d \ge b$. In the first case, since there exists a path from the source which has length at most bc^j and which visits at least i vertices, our j^{th} subtour must visit at least i vertices. This allows us to upper-bound u_i by

$$u_i \quad \le bc^j + 2\textstyle\sum_{l=1}^{j-1} bc^l \quad \le bc^j \left(\dfrac{c+1}{c-1} \right).$$

In the second case, we are similarly guaranteed that our $(j+1)^{st}$ subtour includes at least i vertices, so that

$$u_i \quad \le bc^{j+1} + 2\textstyle\sum_{l=1}^{j} bc^l \quad \le bc^{j+1} \left(\dfrac{c+1}{c-1} \right).$$

In the first case, $U \in [\log_c d, 1]$ and in the second case, $U \in [0, \log_c d)$. Taking the expectation over U,

$$\begin{aligned}
u_i \quad \le \quad & \int_{\log_c d}^{1} bc^j \left(\frac{c+1}{c-1} \right) \mathrm{d}U + \\
& \int_{0}^{\log_c d} bc^{j+1} \left(\frac{c+1}{c-1} \right) \mathrm{d}U \\
= \quad & c^j \left(\frac{c+1}{c-1} \right) \left(\int_{\log_c d}^{1} c^U \mathrm{d}U + c \int_{0}^{\log_c d} c^U \mathrm{d}U \right) \\
= \quad & dc^j \left(\frac{c+1}{\ln c} \right).
\end{aligned}$$

Therefore the ratio between the worst case latency of the i^{th} vertex in our tour and the i^{th} vertex in the optimal tour is bounded by $\frac{c+1}{\ln c}$. This quantity is minimized for $c = 3.59$ for which the approximation ratio turns out to be $\frac{c+1}{\ln c} = c = 3.59$.

4 Finding good k-trees

In this section we present an algorithm, which we will call the constrained k-MST algorithm, which outputs a k-tree including s and t of cost no more than the optimum k-path from s to t, for certain values of k. Thus we are computing a solution to what is essentially the k-MST problem (with the added required vertex t), and comparing its cost to the optimum of a closely related but different problem: the minimum k-path problem.

As one might expect, the algorithm and its analysis take inspiration from algorithms for the k-MST problem used by Blum, Ravi, and Vempala [11] and by Garg [20], based on the algorithm for prize collecting Steiner tree (PCST) problem by Goemans and Williamson [22]. We use the primal-dual schema to solve a Lagrangian relaxation of the problem. Varying the parameter λ in the relaxation gives us a set of trees, with tree T_{k_λ} containing k_λ vertices. The cost of T_{k_λ} will be no more than the cost of the optimal k_λ-path.

In Subsection 4.1, we give an LP relaxation for the k-path problem, its dual, and notation used subsequently. In Subsection 4.2 we describe the algorithm; in 4.3 we bound the cost of the tree it returns in terms of the optimum k-path, and discuss how to interpolate to get such trees for every k, or to obviate the need to do so.

4.1 The k-path problem

Our primal-dual algorithm makes use of the following linear programming relaxation for the problem of finding the minimum k-path from s to t. We associate indicator variables x_e with edges, x_e being 1 if e is on the path and 0 otherwise. Similarly, variables x_v select the $(k-2)$ vertices (other than s and t) on the path. $\delta(S)$ denotes the set of edges with exactly one endpoint in set S.

$$\begin{array}{ll} \text{minimize} & \sum_{e\in E} c_e x_e \\ \text{subject to} & \end{array}$$

$$\begin{array}{rcll} \sum_{e\in\delta(S)} x_e & \geq & 2x_v & \forall S\subseteq V\setminus\{s,t\}, \forall v\in S \\ \sum_{e\in\delta(U)} x_e & \geq & 1 & \forall U\subseteq V : t\in U, s\notin U \\ \sum_{v\in V\setminus\{s,t\}} x_v & \geq & k-2 & \\ x_v & \leq & 1 & \forall v\in V\setminus\{s,t\} \\ x_v & \geq & 0 & \forall v\in V\setminus\{s,t\} \\ x_e & \geq & 0 & \forall e\in E \end{array}$$

It is easy to see that any valid k-path from s to t defines an assignment to the variables satisfying all constraints. Hence the optimum of this linear program is a lower bound on the optimum k-path.

The dual of the above linear program has a variable p, variables p_v for each vertex v, variables $y_{v,S}$ for each $S\subseteq V\setminus\{s,t\}$ and each $v\in S$, and variables $y_{t,U}$ for each $U\subseteq V\setminus\{s\}$ such that $t\in U$. Note that the subscript t in $y_{t,U}$ is redundant, and used only for notational consistency. The dual is as follows.

$$\begin{array}{ll} \text{maximize} & (k-2)p - \sum_{v\in V\setminus\{s,t\}} p_v \\ & + \sum_{U:t\in U, s\notin U} y_{t,U} \\ \text{subject to} & \end{array}$$

$$\begin{array}{rcll} 2\sum_{S\ni v} y_{v,S} + p_v & \geq & p & \forall v\neq t \\ \sum_{S:e\in\delta(S)} \sum_{v\in S} y_{v,S} & & & \\ + \sum_{U:t\in U, e\in\delta(U)} y_{t,U} & \leq & c_e & \forall e\in E \\ p_v & \geq & 0 & \forall v\in V \\ y_{v,S} & \geq & 0 & \forall S\subseteq V\setminus\{s,t\}, \\ & & & \forall v\in S \\ y_{t,U} & \geq & 0 & \forall U\subseteq V\setminus\{s\} : \\ & & & t\in U \end{array}$$

The rather unwieldy edge constraints can by simplified by introducing the "missing variables" $y_{v,S}$ for all $S\subseteq V$ and $v\in S$ which were not defined above, and setting them constantly to zero. For convenience, we also define

$$z_S = \sum_{v\in S} y_{v,S}.$$

(Note that for any set U such that $t\in U$ and $s\notin U$, z_U is just $y_{t,U}$.) The edge constraints can now be rewritten as follows.

$$\begin{array}{rcll} \sum_{S\subseteq V: e\in\delta(S)} z_S & \leq & c_e & \forall e\in E \\ y_{v,S} & = & 0 & \forall S\ni s, \forall v\in S \\ y_{v,U} & = & 0 & \forall U : t\in U, s\notin U, \\ & & & \forall v\in U\setminus\{t\} \end{array} \qquad (1)$$

Definition 1 *A* potential assignment *is a function π from vertices to non-negative reals such that $\pi(s)=0$. We say that π is* feasible *if we can define non-negative variables $y_{v,S}$ for each $v\in V$ and $S\subseteq V$ such that (1) for all vertices v, $\pi(v)\leq\sum_{S\ni v} y_{v,S}$, (2) the edge constraints (Equation 1) are satisfied.*

Our algorithm constructs variables $y_{v,S}$ which yield a feasible potential assignment π. We then use π to analyze the cost of the tree returned by the algorithm.

4.2 The algorithm

We are given as input a graph $G=(V,E)$, a source $s\in V$, a sink $t\in V$, and a parameter λ. The algorithm consists of two phases. In the *growth phase*, we iteratively add edges, paying for them with increases to variables in the dual of the k-path LP. In the *delete phase*, we do some pruning to obtain the final tree.

Growth phase. The state at each iteration is as follows. The algorithm maintains a partitioning of the graph into components which are either *active* or *inactive*. Each vertex v has an associated non-negative budget b_v, which will help pay for further growth of components containing v. Components containing a vertex with positive budget and which do not contain s are active; those whose vertices all have zero budget, and the component containing s, are inactive. To keep track of how much budget has been spent, we employ variables $y_{v,S}$ for each $v\in V$ and $S\subseteq V$.[4]

Initially, each vertex forms its own component and has budget λ, except t, which has infinite budget, and s, which has zero budget. The variables $y_{v,S}$ are all initialized to 0.

At each step, the algorithm picks a small value ϵ and a vertex $v_S\in S$ for each active component S. For the component containing t (if active), the vertex t is always picked; for other components, any vertex $v\in S$ with $b_v>0$ is picked. The algorithm then simultaneously decreases by ϵ the budget of each picked vertex, and increases by ϵ the variables $y_{v_S,S}$ for each active set S

[4] Although there are exponentially many variables $y_{v,S}$, we need only keep track of those that our algorithm makes non-zero, of which there are only polynomially many.

and its picked vertex v_S. Note that this effectively raises each z_S by ϵ.

The value of ϵ is chosen so that it is the smallest for which one of the following occurs:

1. The budget of some picked v_S falls to zero. In this case another vertex $v \in S$ with $b_v > 0$ is picked; if there is no such vertex, S becomes inactive.

2. The variables $y_{v,S}$ increase enough to pay for some edge $e \in E$ joining two components C_1 and C_2; that is, $\sum_{S:e\in\delta(S)} z_S = c_e$. In this case we replace C_1 and C_2 with a new component $C = C_1 \cup C_2 \cup \{e\}$. C becomes inactive if $s \in C$.

The growth phase of the algorithm terminates when all components become inactive.

Delete phase. Let T be the component containing s at the end of the growth phase. Note that T, like all other components, is a tree. We delete all subtrees $S \subseteq T \setminus \{s\}$ such that S formed an inactive component sometime during the growth phase. (Note that sets $S \ni t$ will never qualify.) We repeat this operation until no such subtrees remain, so that every leaf of the pruned tree $T_{k_\lambda} \subseteq T$ was always in an active component until it joined the component containing s.

The algorithm returns the pruned tree T_{k_λ} and the dual variable assignments $y_{v,S}$.

4.3 Analysis

We first define a potential assignment in terms of the dual variables returned by the algorithm:

$$\pi(v) = \sum_{S\ni v} y_{v,S}.$$

Lemma 1 *π is feasible.*

Proof: The variables $y_{v,S}$ returned by the algorithm witness the feasibility of π, as follows. The first feasibility requirement of Definition 1 is satisfied trivially by our definition of π. To see that the edge constraints are fulfilled, note that when an edge's constraint reaches equality, the algorithm merges the components which it joined. After that point, the edge is "buried" in its component, and will never be on the boundary $\delta(S)$ of an active set S, so the variables z_S with $e \in \delta(S)$ will never be raised further. Finally, the variables which are required to be zero are zero throughout the run of the algorithm since the component containing s is always inactive, and t is always picked in the growth phase. ■

Lemma 2 *The nodes outside T_{k_λ} have potential λ, which is the highest among all nodes except t.*

Proof: Note that the potential $\pi(v)$ is exactly the amount of budget that was transferred out of v, so the maximum potential of any node other than t is the initial budget λ, which is never overspent. Consider any node $v \notin T_{k_\lambda}$. It follows that v was part of an inactive component at the end of the algorithm, and hence spent its full budget and has potential λ. ■

Lemma 3 *T_{k_λ} has cost at most $2\sum_{v\in T_{k_\lambda}} \pi(v) - \pi(t)$.*

Proof: We must show that $\sum_{e\in T_{k_\lambda}} c_e \leq 2\sum_{v\in T_{k_\lambda}} \pi(v) - \pi(t)$. The left side of that inequality is equal to

$$\sum_{e\in T_{k_\lambda}} c_e = \sum_{e\in T_{k_\lambda}} \sum_{S:e\in\delta(S)} z_S = \sum_{S\subseteq T_{k_\lambda}} z_S|T_{k_\lambda} \cap \delta(S)|,$$

where the first step follows from the fact that when edges are added in the growth phase of the algorithm, the edge constraints are satisfied with equality for edges in T_{k_λ}. The right side of the inequality is equal to

$$\begin{aligned} 2\sum_{v\in T_{k_\lambda}} \pi(v) - \pi(t) &= 2\sum_{S\subseteq T_{k_\lambda}} \sum_{v\in S} y_{v,S} - \sum_{S\ni t} y_{t,S} \\ &= 2\sum_{S\subseteq T_{k_\lambda}} z_S - \sum_{S\ni t} z_S, \end{aligned}$$

due to our definition of π. So it suffices show that

$$\sum_{S\subseteq T_{k_\lambda}} z_S|T_{k_\lambda} \cap \delta(S)| \leq 2\sum_{S\subseteq T_{k_\lambda}} z_S - \sum_{S\ni t} z_S. \quad (2)$$

We accomplish this by induction on the steps of the growth phase of the algorithm. The base case holds trivially, as all $y_{v,S}$ variables are initialized to 0.

At any step, let N_a and N_i denote current sets of active and inactive components, respectively, that will eventually be part of T_{k_λ}. Let H be a tree whose vertices are $N_a \cup N_i$ and whose edges are the edges of T_{k_λ} that will be added in subsequent stages. For each $S \in N_a$ the algorithm raises z_S by ϵ, causing the left hand side of Equation 2 to increase by $\epsilon\sum_{S\in N_a} \deg(S)$. Since there can be only one active set containing t, it causes the right hand side to increase by $2\epsilon|N_a| - \epsilon$. Thus, dividing by ϵ, it remains only to show that

$$\sum_{S\in N_a} \deg(S) \leq 2|N_a| - 1.$$

Due to the delete phase of the algorithm, inactive components other than the one containing s must be non-leaves of H and so have degree at least 2; s's component has degree at least 1. Thus the sum of the degrees of the components in N_i is at least $2|N_i| - 1$. The sum of the degrees of all of H's vertices is $2|H|-2$, so we have

$$\sum_{S\in N_a}\deg(S)\le(2|H|-2)-(2|N_i|-1)=2|N_a|-1.$$

■

Theorem 1 *There is a feasible*[5] *solution to the dual of the k-path LP with $k=k_\lambda=|T_{k_\lambda}|$, $p=2\lambda$, and objective function value at least the cost of T_{k_λ}. Moreover, $c(T_{k_\lambda})$ is at most the cost of the cheapest k_λ-path.*

Proof: We will set the variables $y_{v,S}$ as returned by the algorithm. By the feasibility of π (Lemma 1), the edge constraints hold. We now show how to pick the remaining variables p_v so that the first constraint in the dual of the LP holds. For $v\notin T_{k_\lambda}$, we have $2\sum_{S\ni v}y_{v,S}=2\lambda=p$ (by Lemma 2), so we can choose $p_v=0$. For $v\in T_{k_\lambda}$, it suffices to set $p_v=p-2\sum_{S\ni v}y_{v,S}$. This produces an objective function value of

$$(k_\lambda-2)p-\textstyle\sum_{v\in V\setminus\{s,t\}}p_v+\sum_{U:t\in U,s\notin U}y_{t,U}$$

$$\begin{aligned}
&= (k_\lambda-2)p-\sum_{v\in T_{k_\lambda}\setminus\{s,t\}}\left(p-2\sum_{S\ni v}y_{v,S}\right)+\pi(t)\\
&= (k_\lambda-2)p-(k_\lambda-2)p+\sum_{v\in T_{k_\lambda}\setminus\{s,t\}}2\pi(v)+\pi(t)\\
&= \sum_{v\in T_{k_\lambda}}2\pi(v)-\pi(t),
\end{aligned}$$

which, by Lemma 3, is an upper bound on the cost of T_{k_λ}. Finally, it follows from weak duality that any k_λ-path from s to t has at least this cost. ■

Thus we have shown how to construct, for some values of k, a k-tree of cost no more than the best k-path. There are two ways we can resolve the issue of not having a tree for every k.

First, we can interpolate between solutions to construct a tree of size exactly k. Using a procedure of Garg [20], this produces a k-tree of cost no more than 2 times the cost of the optimal k-path. Using ideas from Arora and Karakostas [5], we can bring the ratio down to $1+\epsilon$. The improvement to $1+\epsilon$ comes from the idea of "guessing" $O(1/\epsilon)$ points from the optimum path. We omit the details of these methods from this extended abstract.

Second, as we shall show in the next section, for the purposes of approximating the minimum latency tour, it suffices to use only the trees returned by the Lagrangian relaxation, skipping the interpolation.

[5] Here "feasible" refers to satisfying the constraints of the linear program, not our definition of a feasible potential assignment.

5 A few good trees

In this section, we show that the algorithm SP described in Section 3 can do without any interpolated tree. In particular, we show that if we replace each interpolated tree by a *phantom tree*, with cost equal to the corresponding (upper bound on the) cost of the interpolated tree, then the algorithm never uses a phantom tree.

For a given s,t and a parameter λ, the constrained k-MST algorithm of the previous section returns tree T_{k_λ} containing k_λ vertices including s and t and a dual solution $(p^\lambda,\pi^\lambda,y^\lambda)$ to the k_λ-path problem of no smaller cost. Moreover, $p^\lambda=2\lambda$.

Recall the dual of the k-path linear program in Section 4.1. Note that the constraints in this dual linear program depend on s and t but not on k. k appears only in the objective function. For a dual solution (p,π,y), let $\mathrm{cost}_k(p,\pi,y)$ denote the value of the dual objective function. Theorem 1 says that $c(T_{k_\lambda})\le\mathrm{cost}_{k_\lambda}(2\lambda,\pi^\lambda,y^\lambda)$. Now suppose that for arbitrarily close values λ and λ', the trees returned have k_λ and $k_{\lambda'}$ vertices, where $k_\lambda<k<k_{\lambda'}$. Our phantom tree T^k will contain k vertices, and its cost will be set to a linear interpolation of the costs of trees T_{k_λ} and $T_{k'_\lambda}$. More precisely, define

$$\mu=\frac{(k-k_\lambda)}{(k_{\lambda'}-k_\lambda)}.$$

Thus $k=(1-\mu)k_\lambda+\mu k_{\lambda'}$. Then, we will set

$$c(T^k)\quad=\quad(1-\mu)c(T_{k_\lambda})+\mu c(T_{k'_\lambda}).$$

We shall show that the cost of this phantom tree is no more than the cost of the optimal k-path. Since by Theorem 1, the costs of T_λ and $T_{\lambda'}$ are upper bounded by the corresponding dual solutions, it suffices to show that for any k-path P from s to t,

$$c(P)\ge(1-\mu)\mathrm{cost}_{k_\lambda}(2\lambda,\pi^\lambda,y^\lambda)+\mu\mathrm{cost}_{k_{\lambda'}}(2\lambda',\pi^{\lambda'},y^{\lambda'}).$$

Now consider the dual solution $(\hat p,\hat\pi,\hat y)=(1-\mu)(2\lambda,\pi^\lambda,y^\lambda)+\mu(2\lambda',\pi^{\lambda'},y^{\lambda'})$. Since it is a convex combination of dual feasible solutions, it is a feasible solution to the dual linear program. Moreover, note that

$$\begin{aligned}
&\mathrm{cost}_k(\hat p,\hat\pi,\hat y)\\
&= (k-2)\hat p-\sum_{v\in V\setminus\{s,t\}}\hat\pi_v+\sum_{T:t\in T,s\notin T}\hat y_{t,T}\\
&= ((1-\mu)k_\lambda+\mu k_{\lambda'}-2)\hat p-\sum_{v\in V\setminus\{s,t\}}(1-\mu)\pi^\lambda(v)\\
&\quad+\mu\pi^{\lambda'}(v)+\sum_{T:t\in T,s\notin T}(1-\mu)y^\lambda_{t,T}+\mu y^{\lambda'}_{t,T}
\end{aligned}$$

$$
\begin{aligned}
&= (1-\mu)\mathrm{cost}_{k_\lambda}(\hat{p}, \pi^\lambda, y^\lambda) + \mu \mathrm{cost}_{k_{\lambda'}}(\hat{p}, \pi^{\lambda'}, y^{\lambda'}) \\
&\geq (1-\mu)\mathrm{cost}_{k_\lambda}(2\lambda, \pi^\lambda, y, y^\lambda) \\
&\quad + \mu \mathrm{cost}_{k_{\lambda'}}(2\lambda', \pi^{\lambda'}, y^{\lambda'}) - 2\mu k_{\lambda'}(\lambda' - \lambda).
\end{aligned}
$$

Since λ and λ' are arbitrarily close together, this dual solution $(\hat{p}, \hat{\pi}, \hat{y})$ has cost at least the linear interpolation of the two duals. Finally, the cost of any primal solution is more than any dual feasible solution and so the required inequality follows. Thus we have shown that the cost of a phantom tree is no more than the optimal path from s to t containing an equal number of vertices. Hence, for every $k \in [2, \ldots, n]$, and for every $t \in V \setminus \{s\}$, we have generated a real or phantom tree T_k^t such that $c(T_k^t)$ is no more than the cheapest k-path from s to t.

Recall however that in the analysis of the SP algorithm (Section 3), we assumed that the cost of a k-tree was bounded by the optimal k-stroll starting at s. Thus for every k, we need a tree T_k whose cost is no more than the cost of optimum k-stroll starting from s. This is easily achieved by setting T_k to be the minimum cost tree in the set of trees $\{T_k^t : t \in V \setminus \{s\}\}$. Since the optimum k-stroll P_k starting at s ends at some $t^* \in V \setminus \{s\}$, $c(T_k^{t^*}) \leq c(P_k)$ and since we choose T_k to be the cheapest amongst all T_k^t, $c(T_k) \leq c(P_k)$.

Let $\mathcal{T}$ be the set of real trees in $\{T_k : 2 \leq k \leq n\}$ (that is, those returned by the algorithm of Section 4). For any k such that $T_k \notin \mathcal{T}$, we redefine the cost of the phantom tree T_k to be a linear interpolation of the closest trees in $\mathcal{T}$ on either side. It is easy to see that this redefinition only reduces the cost of each phantom tree, and thus we still have $c(T_k) \leq c(P_k)$ (see Figure 1(b)).Also note that there is no loss of generality in assuming that $c(T_{k+1}) \geq c(T_k)$ for every k.

Finally, from this pool of trees, we shall show that if the SP algorithm uses a phantom tree, it can be replaced by a real one. This argument closely follows that used by Archer, Levin and Williamson [3]. Let T_a, T_b and T_c be three consecutive trees used by the algorithm where $a < b < c$ and $c(T_a) < c(T_b) < c(T_c)$. Further let T_b be a phantom tree formed by a linear interpolation of T_{b_0} and T_{b_1}, where $b = (1-\mu)b_0 + \mu b_1$, and $c(T_b) = (1-\mu)c(T_{b_0}) + \mu c(T_{b_1})$. If $a > b_0$, let μ_a be such that $a = (1-\mu_a)b_0 + \mu b_1$, and let $\mu_0 = \max\{0, \mu_a\}$. Similarly, let $\mu_1 = \min\{1, \mu_c\}$, where μ_c is such that $c = (1-\mu_c)b_0 + \mu_c b_1$. Now the cost of the path segment $a \to b \to c$ is given by

$$
\begin{aligned}
&(n - \tfrac{a+b}{2})c(T_b) + (n - \tfrac{b+c}{2})c(T_c) \\
&= (n - \frac{a + (1-\mu)b_0 + \mu b_1}{2})((1-\mu)c(T_{b_0}) \\
&\quad + \mu c(T_{b_1}) + (n - \frac{(1-\mu)b_0 + \mu b_1 + c}{2})c(T_c),
\end{aligned}
$$

which is a quadratic function of μ where the coefficient of μ^2 is $-\frac{b_1 - b_0}{2}(c(T_{b_1}) - c(T_{b_0})) \leq 0$. Thus this function has no local minima and is thus minimized at one of the endpoints of the interval $[\mu_0, \mu_1]$. Therefore b can be replaced by one of a, b_0, b_1, c. Thus we can reduce the number of phantom points by one without increasing the cost of the tour. Repeating this process, we can get a tour which contains no phantom points.

5.1 Running time

A naive implementation of the above algorithm would require $O(n^2)$ k-trees, one for each (t, k) pair. Further, we need λ and λ' in the analysis above to be arbitrarily close to each other. However, assuming integer edge weights, it is easy to show that a separation of $\Omega(\frac{1}{n!})$ suffices, so each of the $O(n^2)$ trees can be computed in $O(n \log n)$ calls to our constrained k-MST algorithm of Section 4.2.

How quickly can we implement a single constrained k-MST computation? As in previous k-MST algorithms, our constrained k-MST algorithm is actually a special case of existing algorithms for the prize collecting Steiner tree (PCST) problem. Since the PCST can be implemented in time $O(n^2)$ [18], we get an overall running time of $O(n^5 \log n)$.

A look at the analysis in Section 3 shows that we only need to construct T_k for $O(\log n)$ different values of k. This immediately improves our running time, and that of the algorithm of [3], by a factor of $n / \log n$.

Moreover, if we are willing to settle for a $(3.59 + \epsilon)$-approximation, we can stop the binary search for k when we find λ and λ' that differ by at most ϵ / n. This helps us replace an $n \log n$ in the running time by $O(\log(n/\epsilon))$. This gives a total of $\tilde{O}(n)$ calls to the PCST, giving a total running time of $\tilde{O}(n^3)$.

5.2 The importance of guesswork

Is it really necessary to try various t? We can show that if running the PCST subroutine with uniform budget λ gives a tree T_{k_λ}, then the cost of T_{k_λ} is no greater than the optimum k_λ-path from s to t, for any $t \in T_{k_\lambda}$. Thus one might surmise that maybe our algorithm would work just as well if we just ran the same k-MST algorithm used in previous minimum latency approximations, and our improvement really comes only from the lower bound.

We now show that this is not the case. Running the algorithm for all possible t does give us better trees. In particular, we show an example on which running the classic k-MST algorithm (i.e. PCST with uniform vertex penalities) would produce a tree whose cost is almost

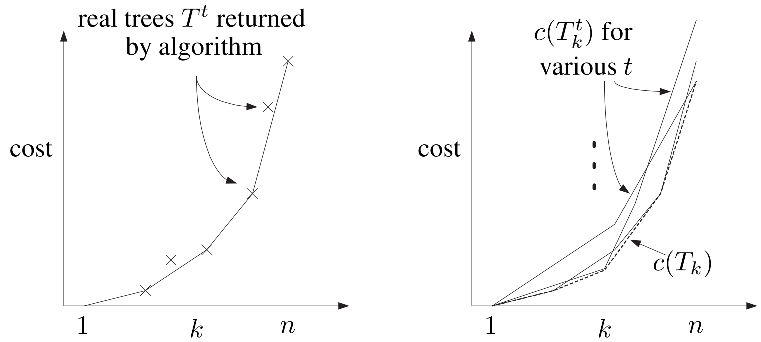

Figure 1. Real and phantom trees. The figure on the left shows phantom trees for a fixed t. The figure on the right shows the final set of trees given to the algorithm SP.

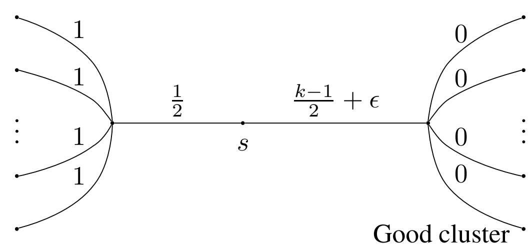

Figure 2. A graph on which our searching over all t helps.

twice the best k-stroll in the graph. On the other hand, our algorithm, for some t, would find a better k-MST.

Consider the graph of Figure 2. There is a unit weighted $(k-2)$ star, with s attached to the root at distance $\frac{1}{2}$. There is also a cluster of $(k-1)$ points at distance zero from each other at distance $\frac{k-1}{2}+\epsilon$ from s, which we call the good cluster.

When the original PCST algorithm is run with budget $\frac{1}{2}$, it returns the star, which is a k-tree of cost $(k-\frac{3}{2})$. On the other hand, our algorithm with a destination t in the good cluster and for $\lambda = 0$ would return the path with the good cluster, which is a k-tree of cost $\frac{k-1}{2}+\epsilon$. The ratio of these two is $2-\frac{1}{k-1}$.

6 Conclusion

We show a simple and efficient 3.59-approximation algorithm to the minimum latency problem. This matches the best known algorithm even on tree metrics, where the k-MST problem is polynomial time solvable. Thus we bypass the factor 2 integrality gap of the k-MST linear program.

The factor of 3.59 comes from the fact that we construct our tour by concatenating smaller tours. This factor is inherent in any such analysis: Goemans and Kleinberg [21] show an example of tours such that the upper bound on the latency of any combination of these tours is τ_n times the sum of the costs of the tours, where the sequence $\tau_n \to \gamma \approx 3.59$. However they do show that this ratio converges rather slowly, e.g. $\tau_{300} < 3.156$. This means that for small graphs, our guarantees are better. We believe that breaking this barrier of γ would involve an approach different than combining small tours.

References

[1] F. Afrati, S. Cosmadakis, C. Papadimitriou, G. Papageorgiou, and N. Papakonstantinou. The complexity of the traveling repairman problem. *Theoretical Informatics and Applications*, 20:79–87, 1986.

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

[3] A. Archer, A. Levin, and D. Williamson. Faster approximation algorithms for the minimum latency problem. Manuscript, 2003.

[4] S. Arora and G. Karakostas. Approximation schemes for minimum latency problems. In *Proceedings of the Thirty-First Annual ACM Symposium on Theory of Computing (STOC'99)*, pages 688–693, New York, May 1999. Association for Computing Machinery.

[5] S. Arora and G. Karakostas. A $(2+\epsilon)$ approximation algorithm for the k-MST problem. In *Proceedings of the Eleventh Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 754–759, N.Y., Jan. 9–11 2000. ACM Press.

[6] G. Ausiello, S. Leonardi, and A. Marchetti-Spaccamela. On salesmen, repairmen, spiders, and other traveling agents. *Lecture Notes in Computer Science*, 1767, 2000.

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

[8] L. Bianco, A. Mingozzi, and S. Ricciardelli. The traveling salesman problem with cumulative costs. *NETWORKS: Networks: An International Journal*, 23, 1993.

[9] A. Blum, P. Chalasani, D. Coppersmith, B. Pulleyblank, P. Raghavan, and M. Sudan. The minimum latency problem. In *Proceedings of the Twenty-Sixth Annual ACM Symposium on the Theory of Computing*, pages 163–171, Montréal, Québec, Canada, 23–25 May 1994.

[10] A. Blum, S. Chawla, D. Karger, T. Lane, A. Meyerson, and M. Minkoff. Approximation algorithms for orienteering and discounted-reward TSP. In *Proceedings of the 44th Annual IEEE Symposium on Foundations of Computer Science*, 2003.

[11] A. Blum, R. Ravi, and S. Vempala. A constant-factor approximation algorithm for the k-MST problem. In *Proceedings of The Twenty-Eighth Annual ACM Symposium On The Theory Of Computing (STOC '96)*, pages 442–448, New York, USA, May 1996. ACM Press.

[12] C. Chekuri and A. Kumar. Maximum coverage problem with group budget constraints and applications. Submitted, 2003.

[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. *Lecture Notes in Computer Science*, 2081, 2001.

[14] J. Fakcharoenphol, C. Harrelson, and S. Rao. The k-traveling repairman problem. In *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms*. ACM/SIAM, Jan. 12–14 2003.

[15] U. Feige, L. Lovász, and P. Tetali. Approximating min-sum set cover. *Lecture Notes in Computer Science: Proceedings of the 5^{th} APPROX*, 2462:94–107, 2002.

[16] E. Feuerstein and L. Stougie. On-line single-server dial-a-ride problems. *Theoretical Computer Science*, 268(1):91–105, Oct. 2001.

[17] M. Fischetti, G. Laporte, and S. Martello. The delivery man problem and cumulative matroids. *Operations Research*, 41:1065–1064, 1993.

[18] H. Gabow and S. Pettie. The dynamic vertex minimum problem and its application to clustering-type approximation algorithms. In *Proceedings 8th Scandinavian Workshop on Algorithm Theory (SWAT'02), LNCS*, pages 190–199, 2002.

[19] H. N. Gabow and S. Pettie. The dynamic vertex minimum problem and its application to clustering-type approximation algorithms. In M. Penttonen and E. M. Schmidt, editors, *SWAT 2002*, volume 2368 of *LNCS*, pages 190–199. Springer, 2002.

[20] N. Garg. A 3-approximation for the minimum tree spanning k vertices. In *37th Annual Symposium on Foundations of Computer Science*, pages 302–309, Burlington, Vermont, 14–16 Oct. 1996. IEEE.

[21] M. Goemans and J. Kleinberg. An improved approximation ratio for the minimum latency problem. In *Proceedings of the Seventh Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 152–158, New York/Philadelphia, Jan. 28–30 1996. ACM/SIAM.

[22] M. X. Goemans and D. P. Williamson. A general approximation technique for constrained forest problems. *SIAM Journal on Computing*, 24(2):296–317, Apr. 1995.

[23] E. Koutsoupias, C. Papadimitriou, and M. Yannakakis. Searching a fixed graph. In *Proceedings of the 23rd International Colloquium on Automata, Languages and Programming*, pages 280–289. Springer-Verlag, 1996.

[24] S. O. Krumke, W. E. de Paepe, D. Poensgen, and L. Stougie. News from the online traveling repairman. *Lecture Notes in Computer Science*, 2136, 2001.

[25] A. Lucena. Time-dependent traveling salesman problem–the deliveryman case. *NETWORKS: Networks: An International Journal*, 20, 1990.

[26] E. Minieka. The delivery man problem on a tree network. *Annals of Operations Research*, 18(1–4):261–266, February 1989.

[27] C. H. Papadimitriou and M. Yannakakis. The traveling salesman problem with distances one and two. *Mathematics of Operations Research*, 18:1–11, 1993.

[28] S. Sahni and T. Gonzales. **P**-complete problems and approximate solutions. In *15th Annual Symposium on Switching and Automata Theory*, pages 28–32, The University of New Orleans, 14–16 Oct. 1974. IEEE.

[29] D. Simchi-Levi and O. Berman. Minimizing the total flow time of n jobs on a network. *IIE Trans.*, 23(3):236–244, September 1991.

[30] R. Sitters. The minimum latency problem is NP-hard for weighted trees. In *IPCO: 9th Integer Programming and Combinatorial Optimization Conference*, 2002.

[31] J. N. Tsitsiklis. Special cases of traveling salesman and repairman problems with time windows. *NETWORKS: Networks: An International Journal*, 22, 1992.

[32] I. R. Webb. Depth-first solutions for the deliveryman problem on tree-like networks: an evaluation using a permutation model. *Transportation Science*, 30(2):134–147, May 1996.

[33] R. J. V. Wiel and N. V. Sahinidis. Heuristic bounds and test problem generation for the time dependent traveling salesman problem. *Transportation Science*, 29(2):167–183, May 1995.

[34] T. Will. *Extremal Results and Algorithms for Degree Sequences of Graphs*. PhD thesis, U. of Illinois at Urbana-Champaign, 1993.

[35] B. Y. Wu. Polynomial time algorithms for some minimum latency problems. *Information Processing Letters*, 75(5):225–229, Oct. 2000.

[36] C. Yang. A dynamic programming algorithm for the travelling repairman problem. *Asia-Pacific J. Operations Research*, 6(10):192–206, 1989.