

Improved Approximation Algorithms for the Minimum Latency Problem via Prize-Collecting Strolls

Aaron Archer*

Anna Blasiak†

Abstract

The *minimum latency problem* (MLP) is a well-studied variant of the *traveling salesman problem* (TSP). In the MLP, the server's goal is to minimize the average latency that the clients experience prior to being served, rather than the total latency experienced by the server (as in the TSP). The MLP sometimes goes by other names, such as the *traveling repairman problem*, or the *deliveryman problem*. Unlike most combinatorial optimization problems, the MLP is NP-hard even on trees (Sitters, 2001). Our main result is an improved approximation algorithm for the MLP on trees, upon which we build improved approximation algorithms for a much wider class of graphs.

The MLP on trees is interesting for several reasons. First, many of the aspects that make the problem difficult on general graphs are already present in the tree case. Second, all existing approximation algorithms for general graphs are built on approximation algorithms for the tree case. Third, there has been no improvement for the tree case since the 3.59-approximation of Goemans and Kleinberg, first introduced 14 years ago in 1996. Fourth, in the intervening period, the best ratio for general metrics has been improved to match the 3.59 for trees (Chaudhuri et al., 2003).

In this paper, we improve the approximation ratio for trees to 3.03. In fact, our 3.03-approximation algorithm works for any class of graphs in which the related *prize-collecting stroll* (PCS) problem is solvable in polynomial time, such as graphs of constant treewidth. More generally, for any class of graphs that admit a Lagrangian-preserving β -approximation algorithm, we can use this algorithm as a black box to achieve a 3.03β -approximation for the MLP. Sadly, this does not immediately improve the ratio of 3.59 for general graphs, because the current best value of β for that case is 2.

One interesting piece of our analysis is the solution of an infinite-dimensional linear program, used to analyze a finite-dimensional factor-revealing linear program (FRLP). We believe that our methods may hold promise for easing the analysis of other FRLPs encountered in the literature.

1 Introduction

Given a metric space M on a set V of n nodes with a root $r \in V$, the famous *traveling salesman problem* (TSP) asks for a tour of minimum total cost that visits every node in V , starting and finishing at r . If the distances between nodes are viewed as travel times for a salesman based at r , then the TSP aims to minimize the total travel time required for the

salesman to visit all of his customers (located at the other nodes in V) and return back home to r . In contrast to this salesman-oriented approach, the *minimum latency problem* (MLP) takes a customer-oriented approach, asking for a tour starting at r and visiting every other node in V in a way that minimizes the average arrival time at all of the customer sites (equivalently, the sum of the arrival times). Formally, the “arrival time” for node v is called the *latency* of v , and is equal to the distance traveled before first arriving at v . The MLP is sometimes called the *traveling repairman problem*, evoking the image of a plumber planning her route so as to minimize the average time her customers have to deal with their leaking pipes. For the weighted version, where each node has a non-negative weight and we wish to minimize the sum of the weighted latencies, Koutsoupias et al. [23] and Ausiello et al. [5] give another motivation. If there is a treasure located at one unknown node of a graph and we set the weight of each node to be the probability that the treasure is located there, then we minimize the expected time to find the treasure by following the tour that minimizes the weighted latency.

In our view, the MLP is just as natural and fundamental a problem as the TSP. It has received plenty of attention, but much less than the TSP. We speculate that this owes primarily to historical accident and inertia, and perhaps to the MLP's having been more resistant to progress.

The MLP is known to be NP-hard [29], and even Max-SNP hard [7, 27] for general metric spaces. While most combinatorial optimization problems are trivial on trees, the MLP is NP-hard even on trees [31]. When we refer to the MLP on a graph, we mean the shortest path metric on a graph with arbitrary costs on its edges. The best approximation ratio known prior to this work, for both the general case and the tree case, is 3.59 [15, 14, 10]. (More precisely, it is $\rho(1)$, where $\rho(a)$ is defined to be the solution to $\rho \log \rho = \rho + a$.) In the present paper, we improve the approximation ratio to $\rho(\frac{1}{3}) \approx 3.03$ for the MLP on trees, the first improvement for this problem in 14 years. This algorithm actually applies to any metric space in which we can solve the related *prize-collecting stroll* (PCS) problem in polynomial time. This is a much larger class than just trees, including all graphs of constant treewidth,

*AT&T Labs – Research, 180 Park Avenue, Florham Park, NJ 07932.
Email: aarcher@research.att.com

†Department of Computer Science, Cornell University, Upson Hall, Ithaca, NY 14853. Email: ablasia@cs.cornell.edu. Supported by an NDSEG Graduate Fellowship, an AT&T Labs Graduate Fellowship, and an NSF Graduate Fellowship. Part of this work was done while employed by AT&T Labs – Research.

for instance. Moreover, for any class of graphs that admit a Lagrangian-preserving β -approximation algorithm for PCS, we can adapt our algorithm to use the PCS algorithm as a black box to achieve a 3.03β -approximation for the MLP.

The MLP on trees is interesting for two main reasons. First, as Kleinberg [22] pointed out, much of the difficulty of the general problem seems to be captured by trees of depth two. For any child v of the root node r , it is clear that the children of v should be visited in order of increasing edge length from v . But the tough question is: each time the tour descends from the root node to v , how many of v 's children should it visit before ascending back to r ?

Second, all of the constant-factor approximation algorithms for the MLP on general graphs are based on approximation algorithms for trees.¹ Blum et al. [7] gave an 8β -approximation for the MLP on general metrics by combining an 8-approximation algorithm for the MLP on trees with any β -approximation for the k -tree problem, used as a black box. Goemans and Kleinberg [14] used a variant of this method to improve the ratio to 3.59β . We henceforth refer to this algorithm as GK. Chaudhuri et al. [10] improved the ratio to 3.59 for general graphs, thereby matching the GK result for trees. Their algorithm is still based largely on the GK algorithm for trees.

Our algorithm builds upon the ideas of Blum et al., GK, and Chaudhuri et al. The Blum et al. and GK algorithms exploit the relationship between the MLP and the k -tree problem (sometimes called k -MST), which asks for the tree of minimum cost spanning k nodes, including the root node r . They generate optimal k -trees for each $k = 2, \dots, n$ (which can be done efficiently since the input graph is a tree), select a particular subsequence of them, and do a depth-first traversal of each selected tree, in order (which we refer to as *concatenating* the trees). By always including the n -tree in the sequence, they ensure that all nodes are visited eventually. Since the cost of the minimum k -tree is a lower bound on the latency of the k^{th} node visited in any tour, the sum of the k -tree costs is a lower bound on OPT . We call this the *tree bound*. They prove that the total latency of their tour is at most 8 (for Blum et al.) or 3.59 (for GK) times the tree bound. The improvement from 8 to 3.59 comes from selecting a different subsequence to concatenate, and a particular depth-first traversal of each tree. Chaudhuri et al. observe that the cheapest *path* visiting k nodes, called the minimum k -stroll, is a stronger lower bound on the latency of the k^{th} node in any tour, so their sum gives a stronger lower bound on OPT . For each k , they generate a k -tree in G whose cost is at most that of the minimum k -stroll, then concatenate a subset of these trees exactly as GK does. Thus,

they are not using the GK black-box result directly,² but they are recycling the major piece.

Our algorithm for the MLP on trees also uses the stroll bound, and starts by generating some (but not all) of the minimum k -strolls (again, this can be done efficiently, since the input graph is a tree). We then select some subsequence of them to concatenate, but both the selection and concatenation processes differ from GK.

When GK concatenates a tree, it must traverse every edge of the k -tree twice, in order to return to the root r to be ready to concatenate the next k -tree. Thus, GK travels twice the length of each concatenated tree. In contrast, one can traverse a stroll directly from root to tail while traveling only the length of the stroll. The problem with ending up at the tail of the stroll is that we cannot immediately traverse the next stroll in the same way. We first observe that we can still traverse a stroll more efficiently than GK traverses a tree, provided that we start at some node inside the stroll. However, there is no reason why the tail of one of our strolls must lie inside the next stroll in our sequence. Therefore, to take advantage of the improved concatenation method, we also need to have some upper bound on the total distance required to move from the tail of one stroll in our concatenation sequence to the nearest node in the next stroll. We guarantee such a bound by using Lagrangian relaxation to generate our original set of strolls by solving the PCS problem, rather than the minimum k -stroll problem. In our view, this is one of the most subtle and interesting parts of our result. We combine these ideas to achieve our approximation ratio of 3.03.

The idea of using Lagrangian relaxation is not new to algorithms for the MLP, but we use it in new ways. Archer et al. [3] used the Lagrangian connection between the k -tree problem and the prize-collecting Steiner tree problem [11, 16] to show that GK doesn't really need to start with minimum k -trees for *all* values of k ; rather it is sufficient to have only the k -trees that are also a minimum prize-collecting tree for some value of the prize parameter λ . The Lagrangian relaxation connection comes in because a *single* optimal prize-collecting tree provides lower bounds on the minimum k -tree for *all* values of k .

We use an analogous connection between k -strolls and PCSs. However, we exploit this connection more deeply than in [3], because we use the PCSs in two orthogonal ways. The first way is analogous to [3]: we bound the cost of our MLP tour not against the stroll bound itself, but rather against the lower envelope of the minimum k -strolls (a weaker bound). Our second critical use is to obtain an appropriate upper bound on the cost to get from the tail of one stroll in our concatenation to the nearest node in the next stroll.

¹It is a common misconception that there is a reduction from the general case to the tree case. This is not true, and we do not mean to imply it here.

²Since the k -tree problem is NP-hard for general graphs, there is no 1-approximation algorithm unless P=NP.

The crucial difficulty in our algorithm and analysis is bounding the total distance required to move from the tail of one stroll in our concatenation sequence to the nearest node in the next stroll. Using the upper bound given by the PCSs wins only half the battle – we also need to pick our strolls more carefully than in previous algorithms. Previous algorithms select the next tree to concatenate based only on its size and cost, and the size of the previous one, irrespective of how the sets of nodes spanned by the previous trees overlap each other. In contrast, we get mileage (quantified in Lemma 2.18) out of the fact that when these sets are not nested, the number of nodes already visited is strictly larger than the size of the previous tree. Without this extra slack, our analysis would not go through. It is our use of Lagrangian relaxation that enables us to prove (in Theorem 2.16) that the extra slack swamps the extra cost incurred by moving from the tail of one stroll to the nearest node in the next stroll.

Our algorithm is not restricted to trees. Rather, it works for any metric in which we can solve the PCS problem in polynomial time. In fact, it is fairly straightforward to extend our algorithm to obtain a 3.03β -approximation on any metric for which we have a Lagrangian-preserving β -approximation algorithm for the PCS problem. For general metrics, this does not immediately improve the ratio of 3.59 because the current best value of β is 2 [10]. However, there are large, important classes of graphs for which $\beta = 1$: we show in Section 8 that this includes all graphs with constant treewidth. Moreover, we emphasize that the Blum et al. and GK black-box lossy reductions from MLP to k -tree came *before* the first constant-factor approximation algorithm for k -tree [13], yet they ultimately led to the current champion MLP algorithm. Thus, we are optimistic that our methods will play a future role in improving the approximation ratio for general graphs.

We wish to highlight one additional part of our analysis, where we solve an infinite-dimensional linear program. Both Blum et al. and GK select trees to concatenate by creating a geometric sequence of bucket breakpoints, and choosing the tree in each bucket with largest cost. Goemans and Kleinberg observe that an even better concatenation sequence could be identified via a shortest path calculation in an associated graph (the *concatenation graph* of Definition 2.6, with $a=1$). Implicit in their discussion is that an alternate way to derive their approximation ratio would be to analyze a certain *factor-revealing linear program* (FRLP), whose solution defines the concatenation graph that produces the worst possible approximation bound. However, since they fortuitously achieved this same bound via other means (i.e., bucketing), they chose not to pursue this avenue. We view the use of the FRLP as a more natural route to prove the bound, and an interesting technical challenge. The FRLP depends on the number of nodes $n = |V|$. For finite n , it is unclear how to find the optimal solution of the

FRLP analytically. Our insight is that by taking the limit as $n \rightarrow \infty$, the FRLP turns into an infinite-dimensional linear program that turns out to be easy to solve. We use this route to prove our Theorem 2.9.

We could have proved Theorem 2.9 by using bucketing similar to [14], but doing the analysis via FRLPs sheds more light on the GK algorithm, and resolves a mystery from a paper by Arora and Karakostas [4]. The analysis of GK in [14] shows that GK obtains a deterministic approximation ratio of 3.59 if the concatenation sequence is chosen via a shortest path calculation in the concatenation graph, and an expected ratio of 3.59 if it is chosen via their bucketing procedure (which applies a random shift to the buckets). Both of these procedures are *adaptive*, depending on the costs of the trees. Our analysis via the FRLP shows that one can select a certain *random* sequence $k_1 < k_2 < \dots < k_i = n$ *before even knowing the input*, and still obtain an approximation ratio of 3.59, in expectation. This helps explain the discovery of Arora and Karakostas where they *fix* a subsequence of k -trees without any knowledge of their costs, and still manage to prove a constant factor of 5.828 for the MLP on trees (Theorem 4.11 of [4]).

The use of FRLPs to help prove approximation ratios was first advocated by Jain et al. [20], although it was implicit in Chvatal’s analysis of the greedy algorithm for set cover [12]. It has since been applied repeatedly in various contexts such as [24, 25, 6, 18, 19, 2]. It can often be difficult to analyze an FRLP, and we believe that our technique of explicitly analyzing the limiting infinite-dimensional FRLP will make this task easier for future researchers encountering FRLPs in other contexts.

This paper is structured as follows. Section 2 gives a simpler version of our main algorithm, yielding an approximation ratio of 3.18. In Section 3, we modify our algorithm (using randomization) to improve the ratio to 3.03 (in expectation). We derandomize it in Section 6. Section 5 discusses how to extend it to a 3.03β -approximation, using a Lagrangian-preserving β -approximation for PCS as a black box. The proof of the key Theorem 2.9 using infinite-dimensional FRLPs appears in Section 4. Section 7 shows how to solve the PCS problem for trees in $O(n)$ time via a simple dynamic program, and how to efficiently compute the entire lower stroll envelope in time $O(n^2)$. Finally, in Section 8 we give an algorithm for computing the optimal PCS in any graph with constant treewidth in $O(n)$ time, and the entire lower stroll envelope on $O(n^2)$ time. This demonstrates that our improved approximation ratio applies to a much larger class of graphs than just trees. The overall running time for the algorithm is $O(n^2)$ in these metrics, being dominated by the time to compute the lower stroll envelope.

2 Approximation Algorithm

In this section, we give a 3.18-approximation algorithm for the MLP on trees, graphs of constant treewidth, and any other graphs where we can solve the related *prize-collecting stroll* problem in polynomial time.

Recall that we are given a finite metric space on a set V of n nodes, plus a root $r \in V$, and we would like to find a tour T with minimum total latency, starting at r and visiting all nodes of V . The latency of a node is the total distance traveled in T before arriving at that node, and the latency of T is the sum of all the node latencies. The latency of r is zero, since we start there. It will be easiest to describe the construction of a tour that may revisit some nodes, but the latency of a node is based only on the first visit. After describing this version of the tour, we can shortcut repeated nodes to arrive at the final tour, which visits each node exactly once. By the triangle inequality, the shortcutting does not increase the latency of any node.

It can be useful to rewrite the total latency as

$$\sum_{i=1}^{n-1} (n-i)c(v_i, v_{i+1})$$

where $c(v_i, v_{i+1})$ is the cost of traveling from the i^{th} node to the $(i+1)^{\text{th}}$ node visited in the tour. This gives a convenient way to amortize the total latency among sections of the tour because we can compute the latency of a section without knowing details of the rest of the tour.

DEFINITION 2.1. Given a tour $T = v_1 \dots v_n$, and a sub-path $W = v_i v_{i+1} \dots v_j$, we define

$$\text{lat}(W) = \sum_{k=i}^{j-1} (n-k)c(v_k, v_{k+1}),$$

and say that W adds $\text{lat}(W)$ to the latency of T . Although $\text{lat}(W)$ depends on where W lies within T , we suppress this dependence in the notation. Whenever we refer to $\text{lat}(W)$, W will be one option for the next segment of the tour that we are in the middle of constructing.

Our algorithm produces a tour by traversing a sequence of strolls. A k -stroll is a path visiting k distinct nodes, starting at the root r . The cost of a stroll S , denoted $c(S)$, is the sum of the distances between each pair of consecutive nodes in the stroll. The minimum k -stroll is the k -stroll with minimum cost. Throughout the paper we will denote the minimum k -stroll as S_k and the minimum $(n-k)$ -stroll as R_k . The letter R is a mnemonic to remind the reader that there are k nodes *remaining*. Note that $S_1 = R_{n-1}$ is the trivial stroll, consisting only of the root node r and having cost 0. Let $V(S)$ denote the set of nodes visited by stroll S . The *tail* of a stroll is the end that is not r .

OBSERVATION 2.2. For the shortest path metric of a tree or any graph with constant tree width, the minimum k -stroll can be computed in polynomial time.

We record this observation here to give an inkling of why k -strolls may be algorithmically useful, but omit the proof because we will not use Observation 2.2 directly.

How do k -strolls relate to the MLP? In any tour T , the initial segment visiting the first k nodes is a k -stroll, and its cost is the latency of the k^{th} node in T . Therefore, the cost of the *minimum* k -stroll is a lower bound on the latency of the k^{th} node in the optimal MLP tour, and summing them gives a lower bound on OPT .

OBSERVATION 2.3. $\sum_{k=2}^n c(S_k) \leq OPT$ where OPT is the total latency of the minimum latency tour.

Our algorithm will traverse a sequence of minimum k -strolls in a certain way, and we will bound its approximation ratio by comparing against the stroll lower bound of Observation 2.3. We now derive the relevant bounds.

DEFINITION 2.4. Given a stroll S and a start node v in S , let $W_{rt}(S, v)$ be the walk that starts at v , follows the stroll backwards to r , then traverses the stroll forwards, ending at the tail. (The subscript rt stands for “root-tail.”) Let $W_{tr}(S, v)$ (“tail-root”) be the walk that starts at v , completes the stroll to the tail, and then follows the stroll backwards, ending at the root.

LEMMA 2.5. Suppose we are partially through a tour, at node v . Let V_0 be the set of nodes left to visit, and let S be a stroll such that $v \in V(S)$. Let $j = |V_0|$, $i = |V_0 - V(S)|$, and $s = c(S)$. Then

$$\begin{aligned} & \min(\text{lat}(W_{rt}(S, v)), \text{lat}(W_{tr}(S, v))) \\ & \leq \frac{1}{2}(\text{lat}(W_{rt}(S, v)) + \text{lat}(W_{tr}(S, v))) \\ & \leq s(j + \frac{i}{2}). \end{aligned}$$

That is, the expected latency of a uniform random choice between a root-tail and tail-root traversal (and hence the better of the two) is at most $s(j + \frac{i}{2})$.

Proof. Let S^r be the first portion of the stroll S starting from r and going to v , and let S^t be the portion of S starting at v and ending at t . Let s_r and s_t be the costs of the paths S^r and S^t respectively. Also, let $k_r = |V_0 \cap V(S^r)|$ and $k_t = |V_0 \cap V(S^t)|$, respectively. In other words, k_r, k_t are the number of nodes in each section that we have not already visited in the tour. Since $v \notin V_0$, we have $k_r + k_t = |V_0 \cap V(S)| = j - i$. First we upper bound $\text{lat}(W_{rt})$.

The traversal from v to r and back to v adds at most s_r latency to the k_r new nodes in S^r , and $2s_r$ latency to

the $i + k_t$ nodes still left to visit. The traversal from v to t adds at most s_t latency to the k_t new nodes in S^t and the i nodes still left to visit. Adding these gives $\text{lat}(W_{rt}) \leq k_r s_r + 2(i + k_t) s_r + k_t s_t + i s_t$. A symmetric calculation gives $\text{lat}(W_{tr}) \leq k_t s_t + 2(i + k_r) s_t + k_r s_r + i s_r$. Thus, the average is

$$\begin{aligned} & \frac{1}{2}(\text{lat}(W_{rt}) + \text{lat}(W_{tr})) \\ & \leq \frac{1}{2}(2k_r(s_r + s_t) + 2k_t(s_r + s_t) \\ & \quad + 3i(s_r + s_t)) \\ & = (k_r + k_t)(s_r + s_t) + \frac{3i}{2}(s_r + s_t) \\ & = (j - i)s + \frac{3i}{2}s = s(j + \frac{i}{2}) \end{aligned}$$

2.1 A Special Case: Nested k -strolls If it so happens that all of the minimum k -strolls are *nested* (i.e., $V(S_i) \subseteq V(S_j)$ for all $i < j$) then Lemma 2.5 is almost enough to give us an improved approximation algorithm. We explain this very special case as a warm-up for the general case.

Recall that R_i denotes the minimum $(n - i)$ -stroll, and let s_i be its cost. Suppose we have a sequence of nested strolls $R_{j_1}, R_{j_2}, \dots, R_{j_m}$, where $j_1 > j_2 > \dots > j_m = 0$ (so $R_{j_m} = S_n$ spans all of V). We can build a tour by first traversing stroll R_{j_1} , from r to its tail v , then traversing whichever of $W_{rt}(R_{j_2}, v)$ or $W_{tr}(R_{j_2}, v)$ has lower latency, and so on. Since the strolls are nested, after visiting the new nodes in $R_{j_{i-1}}$, we are at a node in R_{j_i} and we have exactly j_{i-1} nodes left to visit. Therefore, traversing R_{j_i} adds at most $s_{j_i}(j_{i-1} + \frac{j_i}{2})$ to the latency of the tour, by Lemma 2.5. Because $S_n = R_{j_m}$ is one of the strolls in our sequence, the tour eventually visits every node. Summing the contribution of each stroll upper bounds the latency of our tour by $\sum_{k=1}^m s_{j_k}(j_{k-1} + \frac{j_k}{2})$ (where $j_0 = n$). All that is left is to choose the sequence of strolls to traverse. Choosing the best sequence reduces to a shortest path computation in a certain graph.

DEFINITION 2.6. *Given a real number a and a decreasing sequence $s = (s_0, s_1, \dots, s_{n-1})$ of real numbers, the concatenation graph, $CG(s, a)$ has node set $V = \{0, 1, \dots, n - 1\}$, directed arcs $E = \{(j, i) : j > i\}$, and edge costs $c_{ji} = s_i(j + ai)$ for all $(j, i) \in E$.*

Node j in the concatenation graph represents R_j , the minimum $(n - j)$ -stroll. Any path from node $n - 1$ to node 0 in the graph represents a concatenation of strolls that visits all the nodes in our metric space. If $a = 1/2$ and the strolls are nested, then the cost of any $(n - 1) \rightsquigarrow 0$ path will be an upper bound on the latency of the resulting tour. The

reason for formulating the concatenation graph with a as a parameter is that later we will encounter a concatenation graph with $a = 1/3$; moreover, the Goemans-Kleinberg approximation algorithm uses $a = 1$.

DEFINITION 2.7. *For $a \geq 0$, let $\rho(a)$ be the unique solution to $\rho \ln \rho = \rho + a$.*

DEFINITION 2.8. *Let $SP(s, a)$ denote the shortest $(n - 1) \rightsquigarrow 0$ path in $CG(s, a)$, and $c(SP(s, a))$ its cost.*

THEOREM 2.9. (LENGTH OF SHORTEST PATH) *For all n and all (s_0, \dots, s_{n-1}) , $c(SP(s, a)) \leq \rho(a) \sum_{i=0}^{n-1} s_i$.*

If we apply Theorem 2.9 to $CG(s, 1/2)$ where $s_i = c(R_i)$ and the strolls happen to be nested, then it gives us an approximation ratio of $\rho(1/2) \approx 3.18$, Observation 2.3 and Lemma 2.5.

Goemans and Kleinberg [14] proved the case of Theorem 2.9 when $a = 1$, and their methods can easily be extended to handle all $a \geq 0$. In Section 4, we provide an alternate proof using infinite-dimensional factor-revealing linear programs that lends a fresh perspective on the result, and solves a mystery raised by Arora and Karakostas [4].

2.2 The General Case There is no reason for the optimal k -strolls to be nested, so in general we will need a slightly different algorithm. Rather than consider all minimum k -strolls, we will consider a special subset of minimum strolls that have additional structure. Given a penalty p_v for each node $v \in V$, the *prize-collecting stroll (PCS)* problem asks for the stroll S that minimizes $c(S) + \sum_{v \notin V(S)} p_v$. In other words, it is the stroll that minimizes the cost of the edges traversed plus a penalty incurred for each node not visited by the stroll.³ Throughout this paper, when we refer to the PCS problem, we will mean the version with uniform penalties, i.e., $p_v = \lambda$ for all v . Let $S(\lambda)$ denote the minimum PCS with uniform penalties λ .

The PCS problem is closely related to the minimum k -stroll problem, in that PCS is a Lagrangian relaxation of k -stroll.

OBSERVATION 2.10. *For all $\lambda \geq 0$, $k = 1, \dots, n$, we have $c(S_k) \geq c(S(\lambda)) + \lambda(k - |V(S(\lambda))|)$.*

Proof. Otherwise, S_k would be a better PCS than $S(\lambda)$. ■

Applying Observation 2.10 with $k = |S(\lambda)|$ implies that for all λ , $S(\lambda)$ is the minimum $|V(S(\lambda))|$ -stroll. Thus, the set of optimal PCSs $\{S(\lambda) : \lambda \geq 0\}$ constitute a subset of the minimum k -strolls.

³The *prize-collecting* descriptor originates from an equivalent problem of maximizing the total p_v value collected minus the cost of the edges traversed, but this complementary version of the objective function is more traditional.

Figure 1 graphically illustrates the relationship between the optimal k -strolls and the optimal PCSs. Each possible stroll S in the graph is represented by a dot in the figure, with coordinates $(n - |V(S)|, c(S))$. The minimum k -stroll is the lowest dot with x -coordinate $(n - k)$. The dotted line, of slope $-\lambda$, passes through the point representing $S(\lambda)$, and its height at x -coordinate i represents the lower bound on $s_i = c(R_i)$ given by Observation 2.10. If we consider all of the lower bounds generated for all s_i via all possible values of λ , we get the lower envelope shown by the three solid segments in the figure. The set of optimal prize-collecting strolls, i.e., $\{S(\lambda) : \lambda \geq 0\}$, corresponds to the four corner points of this lower envelope.

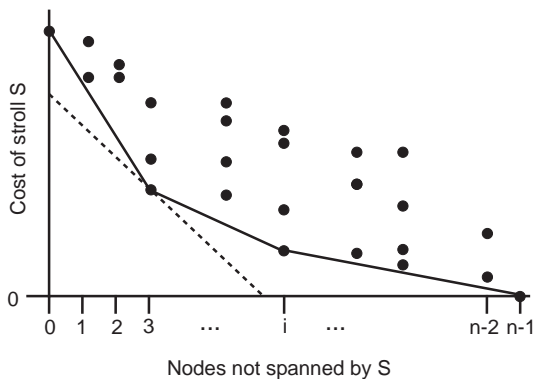


Figure 1: Lower envelope of the k -strolls.

LEMMA 2.11. *For the shortest path metric of any tree or any graph with constant tree width, the lower stroll envelope and the strolls corresponding to its corner points can be computed in polynomial time.*

We prove Lemma 2.11 in Section 8. Before giving our approximation algorithm based on PCSs, we provide the following lemma that is necessary for the algorithm to be well-defined.

LEMMA 2.12. (CORNER POINT ROUNDING) *Given a decreasing, piecewise linear function f , any $a \geq 0$, and any $k \in \{0, \dots, n - 1\}$, the shortest $k \rightsquigarrow 0$ path in graph $CG((f(0), f(1), \dots, f(n - 1)), a)$ uses only corner points of f (except perhaps for k itself).*

If f is the lower stroll envelope (plotted in Figure 1), then the lemma implies that a shortest path will use only optimal PCSs.

Proof. It is enough to show that the first hop in the shortest $k \rightsquigarrow 0$ path must go to a corner point. Since 0 is a corner point, we are fine if the shortest path goes directly to zero. Otherwise, we examine the first two hops: $k \rightarrow j \rightarrow i$

(with $k > j > i$). The total cost of these two hops is $C(j) \stackrel{\text{def}}{=} f(j)(k + aj) + f(i)(j + ai)$. Suppose for contradiction that $(j, f(j))$ is not a corner point, so it lies in the interior of a line segment of slope $-\lambda$, for some $\lambda > 0$.

Let $(l, f(l))$ and $(u, f(u))$ be the corner points sandwiching j (with $l < j < u$). We will show that moving j to either $L = \max(l, i)$ or $R = \min(u, k)$ strictly decreases the cost of the path, yielding a contradiction. If we move j to coincide with i or k , we can then eliminate the self loop, further decreasing the cost of the path. To determine what value of j minimizes $C(j)$, we take the derivative with respect to j : $C'(j) = f'(j)(k + aj) + f(j)a + f(i)$. This is well-defined on the interval $I = [L, R]$. Moreover, since $f'(j) = -\lambda$ is constant on I , C' is decreasing and therefore C is strictly concave on I . Thus, C obtains its minimum at an endpoint of the interval, at a value strictly less than $C(j)$. ■

We now derive a technical corollary of Lemma 2.12 that is crucial for dealing with non-nested strolls.

DEFINITION 2.13. *Let $-\lambda_j^+$ be the slope of the lower envelope immediately to the left of j , and $-\lambda_j^-$ be its slope immediately to the right of j .*

OBSERVATION 2.14. *For each j , $\lambda_j^- \leq \lambda_j^+$, with the inequality being strict iff j is a corner point. If $i < j$ then $\lambda_i^- \geq \lambda_j^+$.*

Observation 2.14 holds because the lower hull is convex. Another way to see this is that when λ increases, the optimal PCS will span at least as many nodes, so $(n - |V(S(\lambda))|)$ decreases.

COROLLARY 2.15. *Suppose the shortest path $k \rightsquigarrow 0$ in $CG(s, a)$ begins with the two hops $k \rightarrow j \rightarrow i$ (where $k > j > i$). Then $C'(j^+)$, the right-hand derivative of C' at j , satisfies $C'(j^+) = -\lambda_j^-(k + aj) + f(j)a + f(i) \geq 0$.*

Proof. Because j is on the shortest path, perturbing j cannot decrease $C(j)$, so $C'(j^+) \geq 0$. ■

Algorithm 1 with $a = \frac{1}{2}$ gives our pseudocode for constructing a 3.18-approximate MLP tour. The algorithm builds on the one for the nested case, with two key differences. Rather than determine a sequence of strolls at the start we do it dynamically: to decide which stroll to traverse next, we determine the number x of nodes left to visit, then traverse the next stroll S on the shortest path from x to 0 in the concatenation graph. In the nested case, x will always be the number of nodes unspanned by the previous stroll. The other difference is that before traversing either $W_{rt}(S, v)$ or $W_{tr}(S, v)$, we may need to move to a node inside S . The main difficulty in the proof is showing that this extra cost is not too large.

Algorithm 1 MLP Approximation Algorithm

- 1: Input: a finite metric space M on a set V of n vertices, and a parameter $a \geq 0$.
 - 2: $f \leftarrow$ lower envelope of optimal PCSs of V on M . // Compute according to Lemma 2.11.
 - 3: $s_i \leftarrow f(i), i = 0, \dots, (n-1)$.
 - 4: $CG \leftarrow CG((s_0, s_2, \dots, s_{n-1}), a)$. // This is the only place where we use parameter a .
 - 5: Compute the shortest path from node i to node 0 in CG for $i \in \{1, \dots, (n-1)\}$.
 - 6: $N(i) \leftarrow$ the first node after i in the shortest path from i to 0.
 - 7: $x \leftarrow n-1$. // x is the number of distinct nodes we have left to visit in the tour.
 - 8: $S' \leftarrow$ the trivial stroll, starting and ending at r .
 - 9: $v \leftarrow$ root node r . // v is the last node we visited in the previous tour.
 - 10: **while** $x > 0$ **do**
 - 11: $S \leftarrow R_{N(x)}$. // By Lemma 2.12, $N(x)$ is a corner point of f , so $R_{N(x)} = S(\lambda_{N(x)}^-)$.
 - 12: $u \leftarrow$ the first node in S that we reach, starting from v and backtracking along S' toward r . // Since r is in both strolls there always exists such a node.
 - 13: Travel from v to u .
 - 14: **if** $\text{lat}(W_{rt}(S, u)) \leq \text{lat}(W_{tr}(S, u))$ **then** Traverse $W_{rt}(S, u)$ **else** Traverse $W_{tr}(S, u)$.
 - 15: $x \leftarrow$ number of distinct nodes still unvisited after traversing S .
 - 16: $v \leftarrow$ last node visited in the traversal of S . // Will be either r or the tail of S
 - 17: $S' \leftarrow S$
 - 18: **end while**
-

THEOREM 2.16. (FIRST MAIN RESULT) *Algorithm 1 with $a = \frac{1}{2}$ gives a $\rho(\frac{1}{2}) \approx 3.18$ -approximation for the MLP in metric spaces for which the PCS lower envelope can be computed in polynomial time.*

Instead of substituting $a = \frac{1}{2}$, we will carry the parameter a throughout the analysis below, so that we can recycle the analysis later in Section 3. Since $\sum s_i$ is a lower bound on OPT it suffices to show that the tour we construct has latency at most $\rho(a) \sum s_i$. We first analyze the case where the PCSs happen to be nested, then the non-nested case.

Proof. [Nested Case] If the PCSs are nested, then after traversing $R_{N(x)}$ we will have exactly $N(x)$ nodes left to visit. Therefore, the algorithm traverses exactly those strolls in the shortest path $SP(s, a)$. Additionally, since our strolls are nested, v will always lie inside our next stroll, so $u = v$ and we incur no extra latency in step 13 of the algorithm. Thus, traversing stroll $R_{N(x)}$ after stroll R_x contributes exactly $c_{x, N(x)}$ to the latency of the tour. By Theorem 2.9,

the cost of our tour is at most $\rho(a) \sum_{i=0}^{n-1} s_i$, as advertised. ■

If the strolls are not nested, then there is an additional contribution to the total latency whenever we must backtrack from the end of our previous stroll S' to a node in the next stroll S that we want to traverse (line 13 of the algorithm). The upside is that when we finish traversing stroll S we will have visited strictly more than $|V(S)|$ nodes because S' contained some nodes that S did not. This benefits us in two ways. First, Lemma 2.5 gives a smaller bound on the cost of our traversal of S . Second, the remaining cost to complete the algorithm is lower, as quantified by the shortest path potentials in CG . We will show that these bonuses more than make up for the extra backtracking cost. That S is an optimal PCS with penalties $\lambda_{N(x)}^-$ is critical here, because it allows us to bound the extra latency incurred in terms of the number of extra nodes we visit.

DEFINITION 2.17. *For $i \in \{0, \dots, (n-1)\}$, let $\pi(i)$ be the cost of the shortest $i \rightsquigarrow 0$ path in $CG(s, a)$.*

LEMMA 2.18. *The shortest path potentials $\pi(\cdot)$ are strictly increasing. In fact, if $0 < x \leq y$, then $\pi(x) \leq \pi(y) - (y-x)s_{N(y)}$.*

Proof. Because the edge in $CG(s, a)$ from i to any node z costs less than that from $(i+1)$ to z , π is increasing.

Observe that the definition of π implies:

$$\begin{aligned} \pi(y) - (y-x)s_{N(y)} &= s_{N(y)}(y + aN(y)) + \pi(N(y)) - (y-x)s_{N(y)} \\ &= s_{N(y)}(x + aN(y)) + \pi(N(y)) \end{aligned}$$

If $N(y) < x$, then the RHS is precisely the cost of the hop $x \rightarrow N(y)$ plus the shortest $N(y) \rightsquigarrow 0$ path in $CG(s, a)$, and is hence an upper bound on $\pi(x)$. If $N(y) \geq x$, then $\pi(N(y)) \geq \pi(x)$, since π is increasing. ■

Before proving Theorem 2.16 in the general case, we provide an informal outline of the argument, which we find useful for understanding the big picture. Each iteration of the while loop essentially corresponds to one hop in $CG(s, a)$. When we find ourselves at node x , we attempt to hop to $N(x)$ by traversing stroll $R_{N(x)}$. However, due to non-nestedness, we may instead land at some node $z < N(x)$. This can only help us, unless we had to incur a backtracking cost before traversing $R_{N(x)}$. In this case, the backtracking cost is at most $\lambda_{N(x)}^- x|P|$ (where y is defined in the proof below), but we save at least $s_{N(x)} a|P|$ in traversing $R_{N(x)}$ and $s_{N(N(x))}|P|$ in $CG(s, a)$ shortest path potential, compared to what we expected. Properties of the lower stroll envelope and CG shortest path as distilled in Corollary 2.15 imply that the savings exceeds the backtracking cost.

Proof. [Theorem 2.16, General Case] Recall that, just prior to each stroll traversal, x denotes the number of nodes left to visit. Let $l(x)$ denote the additional latency incurred by our algorithm from this point on. We prove by induction on x that $l(x) \leq \pi(x)$ (for all values of x encountered by the algorithm). Hence, the total latency of our tour is at most $\pi(n-1)$, which immediately implies an approximation ratio of $\rho(a)$, as before.

The algorithm terminates when $x = 0$, so $l(0) = \pi(0) = 0$. We split the inductive step into two cases. Since we have x nodes left to visit, $S = R_{N(x)}$. In both cases, we need to show $l(x) \leq \pi(x) = s_{N(x)}(x + aN(x)) + \pi(N(x))$.

Case 1: (S contains node v)

The argument here is identical to the nested case.

Case 2: (S does not contain node v)

Let x, v, u, S , and S' be defined as at the end of step 12 in the while loop, where $S' = R_y$ for some $y \geq x$ that is a corner point of f . The latency $l(x)$ incurred by finishing the algorithm consists of three parts: backtracking from v to u (step 13), traversing $R_{N(x)}$ (step 14), and the remaining iterations of the while loop.

We consider the contribution to the latency from the nodes traversed in the next while loop, and then we use the inductive hypothesis to upper bound the contribution of the nodes traversed in the remaining while loops. Let P be the path from v to u in S' , and $|P|$ denote the number of edges in P . Since stroll $S' = R_y$, where y is a corner point of the lower envelope f , we have $S' = S(\lambda)$ for all $\lambda \in [\lambda_y^-, \lambda_y^+]$. If we were to prune away the entire path P from stroll S' (except for node u), the resulting PCS would save $c(P)$ in stroll cost, but incur an extra $\lambda_y^- |P|$ penalty for the $|P|$ extra nodes it is missing. Since S' is already optimal for λ_y^- , we must have $c(P) \leq \lambda_y^- |P|$. Hence, traveling from node v to u contributes at most $\lambda_y^- x |P|$ to the total latency.

Let z be the number of nodes still left to visit after we traverse $S = R_{N(x)}$. There are $N(x)$ nodes not in S , but at least $|P|$ of them were visited when we traversed S' , so $z \leq N(x) - |P|$. Thus, by Lemma 2.5, traversing S adds at most $s_{N(x)}(x + a(N(x) - |P|))$ latency. By induction, the remainder of the algorithm adds at most $\pi(z)$ latency. But Lemma 2.18 implies $\pi(z) \leq \pi(N(x)) - s_{N(N(x))}|P|$.

Summing these three contributions and using the fact that $\pi(x) = s_{N(x)}(x + aN(x)) + \pi(N(x))$ gives

$$\begin{aligned} l(x) &\leq \lambda_y^- x |P| \\ &\quad + s_{N(x)}(x + aN(x) - a|P|) \\ &\quad + \pi(N(x)) - s_{N(N(x))}|P| \\ &= \pi(x) + \lambda_y^- x |P| - a|P|s_{N(x)} - s_{N(N(x))}|P| \end{aligned}$$

We wish to prove $l(x) \leq \pi(x)$, so it suffices to show that $-\lambda_y^- x + as_{N(x)} + s_{N(N(x))} \geq 0$.

Applying Corollary 2.15 to the two hops $x \rightarrow N(x) \rightarrow N(N(x))$ in CG gives $-\lambda_{N(x)}^- (x + aN(x)) + as_{N(x)} +$

$s_{N(N(x))} \geq 0$. Thus, it suffices to show

$$\lambda_y^- x \leq \lambda_{N(x)}^- (x + aN(x)). \quad (2.1)$$

Since $y \geq x > N(x)$, Observation 2.14 gives $\lambda_y^- < \lambda_y^+ \leq \lambda_{N(x)}^-$, implying (2.1) since $x, a, N(x) \geq 0$. ■

3 An improved algorithm

In proving the bound of Theorem 2.16, we made the worst-case assumption that node v might always be the tail of the previous stroll we traversed. However, our traversal can end either at the root or tail of the previous stroll. If we end at the root and have j nodes left to visit, we can simply traverse the next stroll R_i from root to tail and add only $s_i j$ to our latency, which is generally much better than $s_i(j + \frac{i}{2})$.

We improve the algorithm by keeping track of whether we end the stroll at the root or tail. If we end at the tail of the previous stroll S' , then after backtracking along S' to node u , we can follow either walk $W_{rt}(S, u)$ or $W_{tr}(S, u)$. We call the first option the *tail-tail* (TT) traversal, since we start at the tail of S' and end at the tail of S . We call the second option the *tail-root* (TR) traversal. When we finish S' at the root r , we may instead choose the cheaper RT traversal.

We can view Algorithm 1 as choosing randomly between the TR and TT traversal of S at each stage, then derandomizing each traversal choice individually. This no longer works in the presence of the RT option, because an RT or TT traversal precludes the next traversal from being RT. To circumvent this difficulty, we select our sequence of strolls based on the cost of a certain random concatenation, then derandomize the entire concatenation sequence at the very end, in a coordinated fashion.

LEMMA 3.1. *Suppose we are part-way through a random tour at node v , where v is the root with probability $\frac{1}{3}$, and the tail of the previous stroll with probability $\frac{2}{3}$. Let V_0 be the set of nodes left to visit, S be a stroll with cost s , and $v \in S$. Let $j = |V_0|$ and $i = |V_0 - V(S)|$. Let W be the following random traversal: if $v = r$, W is the RT traversal; if v is the tail, then W is the TR traversal or the TT traversal, with probability $\frac{1}{2}$ each. The expected latency of W is at most $s(j + \frac{i}{3})$.*

Proof. If v is the tail then our expected latency is at most $s(j + \frac{i}{2})$, exactly as given in Lemma 2.5. If $v = r$ then the latency is only at most s_j . The expectation is then

$$\frac{2}{3}s(j + \frac{i}{2}) + \frac{1}{3}s_j = s(j + \frac{i}{3}). \quad (3.2)$$

The transition probabilities that determine the traversal W in Lemma 3.1 yield a steady-state distribution of $\frac{1}{3}$ root

and $\frac{2}{3}$ tail. The tour begins at r , which is both the root and tail of the trivial stroll S_1 . Thus, we can declare it to be the root with probability $\frac{1}{3}$ and a tail with probability $\frac{2}{3}$, thereby placing the process directly in its steady-state distribution. Notice that (3.2) is exactly the cost of the edge $j \rightarrow i$ in $CG((s_0, \dots, s_{n-1}), \frac{1}{3})$. Thus, if the optimal prize-collecting strolls happen to be nested, then we immediately get a randomized $\rho(\frac{1}{3}) \approx 3.03$ -approximation algorithm (in expectation), by concatenating strolls as given by the shortest $(n-1) \rightsquigarrow 0$ path in $CG(s, \frac{1}{3})$. The argument is the same as before.

For the general case, we simply run Algorithm 1 with $a = \frac{1}{3}$, and in step 1 we traverse $W_{rt}(S, r)$ whenever $v = r$, and otherwise randomize uniformly between $W_{rt}(S, u)$ and $W_{tr}(S, u)$.

THEOREM 3.2. (MAIN RESULT) *The randomized MLP algorithm described above gives an approximation ratio of $\rho(\frac{1}{3}) \approx 3.03$ (in expectation), for any metric space in which the stroll lower envelope can be computed in polynomial time.*

Proof. Notice that the sequence of strolls chosen by the algorithm is independent of the coin flips. Consequently, the proof is exactly the same as that of Theorem 2.16, with one exception. In the $v \notin S$ case, we need to backtrack only in the event of a TR or TT traversal, so the expected latency from backtracking is at most $\frac{2}{3}\lambda'x|P|$ instead of $\lambda'x|P|$, which only helps. ■

THEOREM 3.3. *The 3.03-approximation algorithm can be de-randomized in an additional $O(n^2)$ time.*

Once the sequence of strolls has been chosen, the traversals can be derandomized using a shortest path calculation in a graph resembling $CG(s, a)$. We defer the proof of Theorem 3.3 to Section 6.

4 Analyzing the concatenation graph using factor-revealing linear programs

Theorem 2.9 states that $c(SP(s, a))$, the cost of the shortest path from node $n-1$ to 0 in $CG(s, a)$, is never larger than $\rho(a) \sum_{j=0}^{n-1} s_j$, even if an adversary chooses n and (s_0, \dots, s_{n-1}) . This statement is purely graph-theoretic, and thus it holds regardless of whether there is an MLP instance where $s_i = c(R_i)$. For any particular choice of s , we can formulate the shortest path via a minimization LP in the standard way, using flow. By taking its dual, we can reformulate it as a maximization LP. Letting the s_i be variables, we see that the worst vector s for a particular value of n (after scaling down s so that $\sum_j s_j = 1$) can be computed as the solution to the following factor-revealing

linear program (FRLP).

$$\text{maximize } \pi_{n-1}$$

subject to:

$$\pi_j \leq \pi_i + (j + ai)s_i \quad \forall 0 \leq i < j \leq n-1 \quad (4.3)$$

$$\sum_{j=0}^{n-1} s_j = 1 \quad (4.4)$$

$$s \geq 0, \pi_0 = 0$$

The $(j + ai)s_i$ term is the cost of the edge from j to i in $CG(s, a)$, and π_i represents the length of the shortest path from i to 0, so (4.3) is the familiar triangle inequality constraint. Let LP_n denote this FRLP, and $\text{val}(LP_n)$ its optimal value. Our goal is to prove that

$$\limsup_{n \rightarrow \infty} \text{val}(LP_n) \leq \rho(a). \quad (4.5)$$

We will actually prove that (4.5) holds with equality, but the inequality is enough to immediately imply Theorem 2.9. Since s_{n-1} appears only in the normalization constraint, the optimal solution always sets $s_{n-1} = 0$. If we set $s_{n-2} = 0$ in LP_{n+1} , it reduces to LP_n . Hence, $\text{val}(LP_n)$ increases monotonically with n . For any finite n , we can solve LP_n numerically, but we want to obtain the limit. This suggests looking at the “limiting” LP as $n \rightarrow \infty$, which will be an infinite-dimensional *continuous* LP.

Since we are interested in obtaining the upper bound (4.5), we need to look at DLP_n , the dual of LP_n :

$$\text{minimize } \nu$$

subject to:

$$\sum_{l:l < n-1} f_{n-1,l} = 1 \quad (4.6)$$

$$\sum_{g:g > i} f_{gi} = \sum_{l:l < i} f_{il} \quad \forall i = 1, \dots, n-2 \quad (4.7)$$

$$\nu \geq \sum_{g:g > i} (ai + g)f_{gi} \quad \forall i = 0, \dots, n-2 \quad (4.8)$$

$$f \geq 0, \nu \text{ free}$$

In DLP_n , the variable f_{ji} denotes a flow on edge (j, i) in graph $CG(s, a)$. Constraint (4.6) says that the total flow out of node $n-1$ must be 1, and constraint (4.7) imposes flow conservation at each intermediate node. Together, (4.6) and (4.7) imply that the flow into 0 is also 1. Constraint (4.8) is the interesting one. It imposes a lower bound on ν in terms of a weighted sum of the edge flows coming into i , where the weight of a flow equals the coefficient of s_i in the cost of the edge in $CG(s, a)$ that is carrying the flow. Since each weight is $\Omega(i)$, the total flow through node i at optimality must be $O(\frac{1}{i})$. Thus, node $n-1$ must spread its out-flow over at least a constant fraction of the other nodes.

In the limit, s and π become functions on the closed interval $[0, 1]$. The continuous LP, CLP, is fairly straightforward:

$$\begin{aligned} & \text{maximize } \pi(1) \\ & \text{subject to:} \\ & \pi(y) \leq \pi(x) + (y + ax)s(x) \quad \forall 0 \leq x < y \leq 1 \quad (4.9) \\ & \int_{[0,1]} s d\mu = 1 \quad (4.10) \\ & s \geq 0, \pi(0) = 0, \end{aligned}$$

where the μ in (4.10) is Lebesgue measure. Constraints (4.9) and (4.10) correspond to (4.3) and (4.4) in LP_n .

The continuous dual LP, CDLP, is a bit trickier. In the limit, f becomes a *unit leftward flow measure* on the product space $(0, 1]^2$ (defined formally by constraints (4.11), (4.12), and (4.14) below). Informally, if A and B are Borel sets with A lying entirely to the left of B , then $f(B \times A)$ is the amount of flow going directly from B into A . Let $\mathcal{T} = \{(x, y) \in (0, 1]^2 : x > y\}$. Then CDLP is:

$$\begin{aligned} & \text{minimize } \nu \\ & \text{subject to:} \\ & f(\{1\} \times (0, 1)) = 1 \quad (4.11) \\ & f([M, 1] \times (m, M)) = f((m, M) \times (0, m]) \quad (4.12) \\ & \quad \quad \quad \forall 0 < m < M < 1 \\ & \nu \mu(I) \geq \int_{(0,1] \times I} (y + ax) df(y, x) \quad (4.13) \\ & \quad \quad \quad \forall \text{ intervals } I \subseteq (0, 1] \\ & f(\bar{\mathcal{T}}) = 0. \quad (4.14) \end{aligned}$$

Constraints (4.11) and (4.12) correspond to constraints (4.6) and (4.7) in DLP_n . Constraint (4.14) says that f is supported on \mathcal{T} ; in other words, all of the flow is moving to the left. This corresponds to the fact that the flow variable f_{ji} exists in DLP_n only when $j > i$. The interesting constraint is (4.13), which corresponds to constraint (4.8) in DLP_n . We cannot write this constraint in terms of a flow density function on $(0, 1]^2$, because such a density may not exist. In fact, we will later discover an optimal solution to CDLP where the flow measure is supported on a set of Lebesgue measure zero, and hence no Radon-Nikodym derivative exists, i.e., f cannot be represented by a density.⁴

Now that we have formulated the continuous FRLPs, we need to relate them to the finite ones. First we relate CDLP to DLP_n .

LEMMA 4.1. Fix n , and let $I_j = (\frac{j}{n}, \frac{j+1}{n}]$ for $j = 0, \dots, n - 1$. If $(f(\cdot), \nu)$ is a feasible solution to CDLP,

⁴For a thorough primer on Radon-Nikodym derivatives, see [28, Ch. 6].

then $((f_{ji}), \nu)$ is a feasible solution to DLP_n with the same objective function value, where $f_{ji} = f(I_j \times I_i)$, and (f_{ji}) is the vector of these flows.

Proof. Since the objective functions of CLP and DLP_n are both just ν , we need only verify that $((f_{ji}), \nu)$ is feasible for DLP_n . Letting $I = (0, \frac{n-1}{n}] = \cup_{j=0}^{n-2} I_j$ and $J = (\frac{n-1}{n}, 1)$, we see that the total flow out of node $n - 1$ is

$$\begin{aligned} \sum_{l:l < n-1} f_{n-1,l} &= \sum_{l:l < n-1} f(I_{n-1} \times I_l) \\ &= f(I_{n-1} \times I) \\ &= f(\{1\} \times I) + f(J \times I) \\ &= f(\{1\} \times I) + f(\{1\} \times J) \quad (4.15) \\ &= f(\{1\} \times (0, 1)) = 1, \end{aligned}$$

where (4.15) follows from the flow conservation constraint (4.12), the last equality follows from (4.11), and the others follow from the definition of f_{ji} and the additivity of measures. Hence, constraint (4.6) is satisfied. The discrete flow conservation constraints (4.7) follow directly from the continuous ones (4.12). Finally, constraint (4.13) with $I = I_i$ implies

$$\begin{aligned} \nu &\geq \frac{1}{\mu(I_i)} \int_{(0,1] \times I_i} (y + ax) df(y, x) \\ &\geq n \sum_{j:j > i} \int_{I_j \times I_i} (y + ax) df(y, x) \quad (4.16) \end{aligned}$$

$$\begin{aligned} &\geq n \sum_{j:j > i} \int_{I_j \times I_i} (\frac{j}{n} + a \frac{i}{n}) df(y, x) \quad (4.17) \\ &= \sum_{j:j > i} (j + ai) f(I_j \times I_i) = \sum_{j:j > i} (j + ai) f_{ji}, \end{aligned}$$

where (4.16) holds because $\mu(I_i) = \frac{1}{n}$ and f is supported on \mathcal{T} , and (4.17) holds because $x > \frac{i}{n}$ for all $x \in I_i$ (and similarly for y and I_j). ■

Now we relate CLP to LP_n . It is easy to show that every feasible solution for CLP is dominated by one in which $s(\cdot)$ is continuous and monotone decreasing on $[0, 1]$, while $\pi(\cdot)$ is continuous (and monotone increasing). Thus, we need worry about projecting only solutions of this form.

LEMMA 4.2. Suppose $(s(\cdot), \pi(\cdot))$ is feasible for CLP, where both functions are continuous and $s(\cdot)$ is decreasing. Let $s_i = s(i/n)$, $\pi_i = \pi(i/n)$, and $N_n = \sum_i s_i$. Then $((\frac{s_i}{N_n}), (\frac{\pi_i}{N_n}))$ is a feasible solution for LP_n , and its value approaches $\pi(1)$ as $n \rightarrow \infty$.

Proof. The solution $((s_i), (\pi_i))$ is feasible for the triangle inequality constraints (4.3), because these are simply constraints (4.9) restricted to $x, y \in \{0, \frac{1}{n}, \dots, \frac{n-1}{n}\}$. Dividing by N_n satisfies the normalization constraint (4.4),

yielding a solution to LP_n of value $\pi(\frac{n-1}{n})/N_n$. Since monotone functions are Riemann integrable and $N_n = \sum_i s_i$ is just the upper Riemann sum approximation of $\int_0^1 s(x)dx = \int_{[0,1]} s d\mu = 1$, we have $N_n \rightarrow 1$ as $n \rightarrow \infty$. Thus, in the limit, our projected primal solution has value $\lim_{n \rightarrow \infty} \pi(\frac{n-1}{n})/N_n = \pi(1)$, matching the value of our solution to CLP. ■

Now we look for an optimal solution to CDLP. Solving LP_n numerically for some small values of n (like $n = 100$) suggests that the optimal dual solution is a “leapfrogging flow.” That is, node $n - 1$ spreads its unit of out-flow over nodes $n - 2, \dots, i$, for some i , in such a way that constraint (4.8) is tight for each $i' > i$. Now, since node $n - 2$ receives some flow from $n - 1$, it needs to send some flow leftward to satisfy flow conservation. It picks up where node $n - 1$ left off, first adding flow to node i until constraint (4.8) is tight for i , then possibly letting some flow spill over to $i - 1$. We then continue propagating flow from node $n - 3$, and so on. The total flow through each intermediate node i increases as i decreases, because constraint (4.8) is tight for each i and the $(j + ai)$ multiplier in this constraint is decreasing as i decreases (since $a > 0$, and as i decreases, the nodes j sending flow into i are decreasing). Thus, aside from node $n - 1$, no node spreads its out-flow over more than 2 receiving nodes. This suggests that in the limit, each point x in $(0, 1)$ has a flow density $d(x)$ such that the total amount of flow going through a small interval I around x is approximately $d(x)$ times the width of I , each point x receives all of its flow density from a single point $h(x) > x$, $h(\cdot)$ is increasing and $d(\cdot)$ is decreasing. Let $h_1 = \inf\{x : h(x) = 1\}$. Under these assumptions, the constraints in CDLP become

$$\int_{(x, h(x))} d(x) d\mu = 1 \quad \forall x \in (0, h_1) \quad (4.18)$$

$$\nu \geq (h(x) + ax)d(x) \quad \forall x \in (0, 1). \quad (4.19)$$

If infinite-dimensional LPs acted like finite-dimensional ones, then since (4.19) is the constraint corresponding to the primal “variable” $s(x)$ and we expect $s(x) > 0$ for all $x < 1$, complementary slackness would imply that (4.19) should be tight for all $x \in (0, 1)$. However, the strong duality theorem does not hold for all infinite-dimensional LPs [1]. In particular, it happens to fail for this one, so we do not actually need to make (4.19) tight over the entire interval.

However, it is still good intuition that we should make (4.19) tight wherever we can, and in fact it is easy to do so on $(0, h_1)$. If we fix some $b > 1$ and set

$$h(x) = \min(bx, 1), \quad (4.20)$$

which implies $h_1 = \frac{1}{b}$, then $(h(x) + ax) = (a + b)x$ on $(0, h_1)$, so taking $d(x) \propto \frac{1}{x}$ makes the RHS of (4.19) constant on $(0, h_1)$. It is decreasing for $x \in [h_1, 1]$, so those

values of x do not interfere with the feasibility of ν . Plugging into (4.18) to solve for d 's constant of proportionality, we find that

$$d(x) = \frac{1}{x \ln b}, \quad (4.21)$$

and hence the smallest value we can take for ν is $\frac{b+a}{\ln b}$. By calculus, we find that the optimal choice for b is $\rho(a)$, which also implies $\nu = \rho(a)$. We have thus proven:

THEOREM 4.3. *The optimal value of CDLP is at most $\rho(a)$.*

Applying Lemma 4.1, we get Corollary 4.4, which, by the initial discussion in this section, implies Theorem 2.9.

COROLLARY 4.4. *For each n , the optimal value of DLP_n (and, by weak duality, LP_n) is at most $\rho(a)$.*

We will refer to the flow measure above as f^* , and will prove presently that it is indeed optimal. If we could exhibit a feasible solution to CLP of value $\rho(a)$, it would prove that f^* really is optimal for CDLP. However, it turns out that there is no optimal solution to CLP, so we will instead exhibit a family of solutions with values arbitrarily close to $\rho(a)$.

Fix any $\epsilon > 0$. Let $s_\epsilon(x) = \min(\frac{1}{x}, \frac{1}{\epsilon})$ and $N_\epsilon = \int_0^1 s_\epsilon(x) dx = 1 + \ln \frac{1}{\epsilon}$. Let $\pi_\epsilon(x) = \rho(a) \ln \frac{x}{\epsilon} - a$ for $x \in [\epsilon, 1]$ and $\pi_\epsilon(x) = 0$ for $x \in [0, \epsilon)$. Using the fact that the expression $a + z - \rho(a) \ln z$ attains a minimum of zero (at $z = \rho(a)$), it is easy to verify that $(\frac{s_\epsilon(\cdot)}{N_\epsilon}, \frac{\pi_\epsilon(\cdot)}{N_\epsilon})$ is a feasible solution to CLP. This solution has value

$$\frac{\pi_\epsilon(1)}{N_\epsilon} = \frac{\rho(a) \ln \frac{1}{\epsilon} - a}{1 + \ln \frac{1}{\epsilon}} \rightarrow \rho(a) \text{ as } \epsilon \rightarrow 0.$$

Thus, this family of primal solutions proves that the dual solution we exhibited above is indeed optimal. Notice that as $\epsilon \rightarrow 0$, the measure induced on $[0, 1]$ by integrating s_ϵ converges to a point mass at 0.

THEOREM 4.5. *The optimal value of CDLP is exactly $\rho(a)$.*

Goemans and Kleinberg, in effect, proved an upper bound of $\rho(1)$ for LP_n with $a = 1$, but they did so via their random bucketing algorithm, which does not give a solution to DLP_n [14]. They proved their analysis was tight by exhibiting a family of solutions to LP_n , with values approaching $\rho(1)$. Their solution is essentially what one gets from projecting our solution $s_{1/n}(\cdot)$, in the special case when $a = 1$.

We asserted above that no optimal solution to CLP exists. Here is a sketch of the proof. It is possible to perturb the solution f^* to a different flow measure \hat{f} in such a way that (\hat{f}, ν) is still feasible for CDLP (and hence is still optimal), but such that (4.13) is loose for every interval I , although it is tight in the limit for intervals approaching

0. Suppose there were an optimal solution $s^*(\cdot)$ to CLP.⁵ The continuous version of complementary slackness says that the optimal primal and dual solutions must satisfy complementary slackness almost everywhere. But since dual constraint (4.13) is tight nowhere, our hypothetical optimal primal solution $s^*(\cdot)$ must be zero almost everywhere, which contradicts (4.10).

In light of this argument, it is logical that the family of solutions $s_\epsilon(\cdot)/N_\epsilon$ converges to a point mass at zero, because the complementary slackness conditions tell us that this is the only “solution” that would be complementary slack with \hat{f} . The problem is that this limiting primal “solution” is not in our domain of feasible solutions.

Another interesting note is that if we apply the continuous analog of the random bucketing algorithm of Goemans and Kleinberg to the primal solution $s_\epsilon(\cdot)$, the flow measure that results matches f^* , except of course on $(0, 1] \times (0, \epsilon]$.

Arora and Karakostas gave a 5.828-approximation algorithm for the MLP on trees, which involves concatenating a *fixed* sequence of trees, the first of which contains more than $n/2$ nodes [4]. They remarked that this result “seems to go against the received intuition of ‘visit the nodes closest to the start node first’ ” [4, p. 1319]. This used to be mysterious, but viewing their result through the lens of CDLP now clarifies things. Our optimal dual flow measure f^* can be viewed as a random path that jumps from 1 to a random point $x_0 \in [-\frac{1}{\rho(a)}, 1)$ (with probability density $d(x)$), then proceeds to make an infinite sequence of further deterministic hops, x_1, x_2, \dots , where $x_i = x_0 \rho(a)^{-i}$. Calculating $E[x_0]$ gives $\frac{\rho(a)-1}{\rho(a)+a}$. For $a = 1$, $E[x_0] \approx 0.564$. Thus, the first tree chosen by our random path would, on average, contain about $0.436n$ nodes.

Thus, the random path represented by the flow measure f^* shares some of the salient aspects of both the adaptive GK random bucketing algorithm and the Arora-Karakostas fixed concatenation sequence. Like Arora-Karakostas but unlike GK, our path is oblivious to s . Moreover, the GK random bucketing algorithm reduces to our path on worst-case inputs. Hence, our non-adaptive random path can be viewed as a common thread tying together the seemingly disparate GK and Arora-Karakostas algorithms.

5 MLP approximations from Lagrangian-preserving PCS approximations

Our Theorem 3.2 can be extended to apply to any metric space, with a somewhat weaker approximation guarantee. In particular, Theorem 5.3 shows how to transform any “pinpointable” Lagrangian-preserving β -approximation algorithm for PCS into a 3.03β -approximation for the MLP. The best Lagrangian-preserving β -approximation algorithm

⁵Here, we are implicitly taking the π function to be the shortest path potential corresponding to $s^*(\cdot)$.

for general metric spaces has $\beta = 2$, and appears in Chaudhuri et al. [10], the same paper that gives a 3.59-approximation algorithm for the MLP on general metrics. Unfortunately, our method thus gives a 6.06-approximation for general metrics, which does not improve on the 3.59. However, just as there are large classes of metrics for which $\beta = 1$ (i.e., on which PCS is solvable in polynomial time; see Section 8), there may be some intermediate classes of metrics that admit some $\beta \in (1, 1.18)$, in which case Theorem 5.3 would give an improvement.

The basic idea of our 3.03β -approximation algorithm is to construct a β -approximation of the lower stroll envelope, and run the original algorithm using its corner points. To do this, we need corresponding primal solutions corresponding to the corner points, and moreover we need them to satisfy a “tail” property that will allow us to properly bound the backtracking cost from step 13 of Algorithm 1. Satisfying the tail property will require doing a bit of surgery on the strolls and some modification of the approximate lower stroll envelope to accommodate the surgery.

DEFINITION 5.1. *A stroll satisfies the α -tail property if, for each integer z , pruning the last z nodes in the stroll decreases the stroll cost by at most αz .*

Note that any optimal PCS with penalties λ satisfies the λ -tail property, but this need not be the case for β -approximation strolls. Also, notice that the α -tail guarantee becomes stronger as α decreases.

The following definition encapsulates exactly what we need from the lower envelope and corresponding corner point strolls to extend our algorithm.

DEFINITION 5.2. *Given a finite metric space M on a set V of n nodes, with root node r , we define an anchored β -envelope to be a piecewise-linear convex function f such that $f(k)$ is a lower bound on the minimum $(n - k)$ -stroll (for $k = 0, \dots, (n - 1)$), and for each corner point k of f there exists:*

- an $(n - k)$ -stroll R_k with cost at most $\beta f(k)$, and
- R_k satisfies the $(\beta\lambda_k^+)$ -tail property, where $-\lambda_k^+$ is the slope of f just to the left of corner point k .

The strolls R_k corresponding to the corner points k are called anchoring strolls for the envelope f .

We previously defined λ_k^+ and λ_k^- in terms of the slope of the lower stroll envelope at k (Definition 2.13). Our redefinition of these symbols inside Definition 5.2 coincides with the original definition when $\beta = 1$. Similarly, in Section 2 we defined R_k to be the minimum $(n - k)$ -stroll, although we actually used only the ones corresponding to corner points of the lower stroll envelope. We can think

of these as being the anchoring strolls of the anchored β -envelope with $\beta = 1$. Viewed in this way, the notation is consistent. For the duration of this section, we will re-define R_j , λ_j^- and λ_j^+ with respect to the anchored β -envelope currently under consideration, rather than the lower stroll envelope.

Notice that we require our anchoring stroll at corner point k to satisfy only the $\beta\lambda_k^+$ -tail property, not the stronger $\beta\lambda_k^-$ -tail property. In the $\beta = 1$ case, the anchoring stroll at corner point k is the *optimal* PCS for every penalty parameter in $[\lambda_k^-, \lambda_k^+]$, so it actually satisfies the stronger λ_k^- -tail property.

If we plot an anchored β -envelope f on top of Figure 1, it would lie on or below the lower stroll envelope, and its corner points may have different x-coordinates than the corner points of the lower stroll envelope. For each corner point of f , the anchoring stroll would appear as a dot with the same x-coordinate and whose y-coordinate is within a factor of β of the corner point.

3.03 β -approximation algorithm Compute an anchored β -lower envelope f and its anchoring strolls. Run the 3.03-approximation algorithm from Section 3, using f rather than the actual lower stroll envelope, and its anchoring β -approximate corner point strolls and their costs instead of the optimal PCSs.

THEOREM 5.3. *Given a finite metric space M on a set V of n nodes, with root node r , for which we can compute an anchored β -lower envelope and its anchoring strolls in polynomial time, the previous algorithm gives a 3.03 β -approximation to the MLP.*

Proof. If our proof of Theorem 3.2 did not use the structure of the PCS at all, then it is easy to see that if we just pretend the lower bound curve represents our actual stroll costs, then the analysis extends immediately when we multiply all the costs by β . The only part of the proof that uses the structure of the stroll is the piece where we bound the cost of backtracking for non-nested strolls. In that part, we had just finished traversing stroll R_y for some $y \geq x$, and were backtracking along it in preparation for traversing stroll $R_{N(x)}$. We used the fact that R_y satisfied the λ_y^- -tail property; if we instead use the weaker λ_y^+ -tail property, the proof of Theorem 2.16 goes through as-is. Since we are now allowing all costs to be blown up by a factor of β , it suffices for R_y to satisfy the $\beta\lambda_y^+$ -tail property, which is what the anchored β -envelope gives us. ■

Now we discuss how to use Lagrangian-preserving approximation algorithms to produce anchored β -envelopes. We begin with some definitions. If we have a PCS algorithm \mathcal{A} , let $\mathcal{A}(\lambda)$ denote the stroll output by \mathcal{A} when the penalties are set to λ .

DEFINITION 5.4. *A PCS algorithm \mathcal{A} is a Lagrangian-preserving β -approximation algorithm if, for every $\lambda \geq 0$, it outputs a solution $\mathcal{A}(\lambda)$ and a lower bound $LB(\lambda)$ that satisfy*

$$c(\mathcal{A}(\lambda)) + \beta\lambda(n - |V(\mathcal{A}(\lambda))|) \leq \beta LB(\lambda), \quad (5.22)$$

where $LB(\lambda)$ is a lower bound on the cost of the optimal PCS solution with penalties λ .

Lagrangian-preserving is defined this way because it leads to an analog of Observation 2.10.

OBSERVATION 5.5. *Suppose \mathcal{A} is a Lagrangian-preserving β -approximation algorithm. Let $\lambda \geq 0$, and $\overline{LB}(\lambda) = LB(\lambda) + \lambda(|V(\mathcal{A}(\lambda))| - n)$. Then $c(\mathcal{A}(\lambda)) \leq \beta\overline{LB}(\lambda)$ and for all $k \in \{1, \dots, n\}$,*

$$c(S_k) \geq \overline{LB}(\lambda) + \lambda(k - |V(\mathcal{A}(\lambda))|). \quad (5.23)$$

Proof. Rearranging (5.22) gives

$$c(\mathcal{A}(\lambda)) \leq \beta(LB(\lambda) + \lambda(|V(\mathcal{A}(\lambda))| - n)) = \beta\overline{LB}(\lambda),$$

which proves the first part. For the second part, we start with the definitions of $LB(\lambda)$ and $\overline{LB}(\lambda)$, then apply Observation 2.10:

$$\begin{aligned} LB(\lambda) &\leq c(S(\lambda)) + \lambda(n - |V(S(\lambda))|) \\ \overline{LB}(\lambda) &\leq c(S(\lambda)) + \lambda(|V(\mathcal{A}(\lambda))| - |V(S(\lambda))|) \\ &\leq c(S_k) + \lambda(|V(\mathcal{A}(\lambda))| - k) \end{aligned}$$

which rearranges to (5.23). ■

In other words, when we run \mathcal{A} with penalties λ , the algorithm outputs a stroll S of some size $i = |V(S)|$ and a number $\overline{LB}(\lambda)$ that is a lower bound on $c(S_i)$. The stroll S is a β -approximate i -stroll, because it is within a factor of β of the lower bound $\overline{LB}(\lambda)$. Moreover, the lower bound on $c(S_i)$ extends to a lower bound on $c(S_k)$ for all k , via a line of slope λ . If we plot the x-coordinate of stroll S as $(n - |V(S)|)$, then the lower bound line has slope $-\lambda$, analogous to the dotted line in Figure 1, but likely lying strictly below the lower stroll envelope.

Consider what happens to $\mathcal{A}(\lambda)$ as λ ranges continuously over some interval. For some sub-intervals, $\mathcal{A}(\lambda)$ is constant, but at some values of λ , the solution changes. We refer to such a value of λ as a *critical value*. When λ is precisely at a critical value λ_c , then the solution returned by \mathcal{A} depends on how the algorithm chooses to break ties. If it chooses to always break ties as though $\lambda = \lambda_c^-$ (i.e., infinitesimally smaller than λ_c), then it will return the solution, denoted $\mathcal{A}(\lambda_c^-)$, corresponding to the interval immediately to the left of λ_c on the real number line; if it breaks ties as though $\lambda = \lambda_c^+$, then it returns the solution, denoted $\mathcal{A}(\lambda_c^+)$, corresponding to the interval immediately to the right of λ_c .

DEFINITION 5.6. With respect to a PCS algorithm \mathcal{A} , a critical value λ spans the interval $[i, j]$ if $|V(\mathcal{A}(\lambda^+))| = n - i$ and $|V(\mathcal{A}(\lambda^-))| = n - j$ (where $i, j \in \mathbb{Z}^+$ and $0 \leq i < j \leq (n - 1)$). The critical value spans k if it spans some interval $[k_L, k_R]$ such that $k_L < k \leq k_R$.

Every approximation algorithm must exhibit similar behavior as λ approaches 0 or ∞ . When $\lambda = 0$, the optimal PCS is just the trivial stroll, spanning only the root r and having cost zero. Every approximation algorithm must also produce this zero-cost solution, or else its approximation ratio would be unbounded. As $\lambda \rightarrow \infty$, every approximation algorithm must span all nodes, in order to avoid paying even a single λ penalty, which would destroy the approximation ratio. Thus, every $k \in \{1, \dots, (n - 1)\}$ is spanned by some critical value of λ .

DEFINITION 5.7. An algorithm for PCS is pinpointable if, for each $k \in \{1, \dots, (n - 1)\}$, we can find a critical value of λ that spans k , in polynomial time.

Although it is difficult to imagine a PCS algorithm that is not pinpointable, since we are treating the Lagrangian-preserving β -approximate PCS algorithm as a black box, we need to add pinpointability as another technical condition. However, we now demonstrate why you get pinpointability for free, with just about any algorithm you can imagine.

Finding a critical value of λ that spans a given k can usually be accomplished using either Megiddo's parametric search technique [26], or ordinary binary search in conjunction with the method of continued fractions (see e.g., [30, 17]). Megiddo's technique involves running the algorithm with λ explicitly treated as a variable. The method works as long as every comparison involving λ can be written as testing the non-negativity of an affine function of λ . This condition is satisfied by every prize-collecting algorithm that we are aware of for any problem (not just PCS algorithms). The method of continued fractions works as long as we can prove that the critical value we are searching for is a rational number with denominator at most $2^{\text{poly}(|\mathcal{I}|)}$, where $|\mathcal{I}|$ is the size of the instance. Again, every prize-collecting algorithm we have ever met satisfies this condition as well.

Now we are ready for the main technical result of this section.

THEOREM 5.8. Given any pinpointable β -approximation algorithm for PCS, we can construct an anchored β -lower envelope and its anchoring strolls in polynomial time.

Proof. We construct the β -lower envelope and the corresponding corner point strolls using the pinpointable PCS algorithm in a black-box fashion.

- For each k , find a critical λ that spans k , along with corresponding strolls visiting $n - k_L$ and $n - k_R$

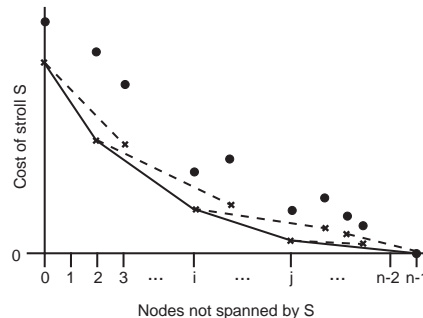


Figure 2: Intervals of lower bounds generated by a β -approximate Lagrangian preserving PCS algorithm, along with their corresponding strolls.

nodes with costs $c(k_L)$ and $c(k_R)$ respectively, and the corresponding lower bounds $LB_L = \overline{LB}(\lambda^+)$ and $LB_R = \overline{LB}(\lambda^-)$. The two strolls give lower bound lines of slope $-\lambda$ going through point (k_L, LB_L) and (k_R, LB_R) , respectively. The lines are parallel, so we pick the better (higher) one, and we get an interval $[k_L, k_R]$ with lower bounds and two corresponding strolls of sizes $(n - k_L)$ and $(n - k_R)$ whose costs are within a factor β of the lower bounds. If we do this for all k , we generate a collection of line segments that cover the line from 0 to n because $n - k_L > n - k$. Figure 2 depicts the segments of lower bounds as dotted lines, marks each segment endpoint by an x , and depicts each corresponding stroll as a dot directly above its x , within a β factor vertically.

- Take the lower convex hull of all of these segment endpoints. Observe that every corner point of the convex hull is a segment endpoint, and hence has a corresponding stroll directly above it, within a factor of β in cost. Figure 2 depicts the lower hull as a solid polyline. At this point, we have all the elements of an anchored β -envelope, except that our anchoring strolls may not satisfy the appropriate tail property.
- We now move from left to right, amending the strolls and the lower hull so that the strolls achieve the $\beta\lambda_k^+$ -tail property. We start with the least corner point k strictly to the right of 0. Let R_k be the corresponding stroll and let z^* be the largest z such that pruning the last z nodes decreases the cost of R_k by at least $\beta\lambda_k^+ z$. Let the stroll R_{k+z^*} be the result of pruning the last z^* nodes from R_k . Notice that R_{k+z^*} satisfies the $\beta\lambda_k^+$ -tail property, by construction. Plot the point corresponding to our new stroll R_{k+z^*} . It is z^* units to the right and more than $z^*\beta\lambda_k^+$ units down from

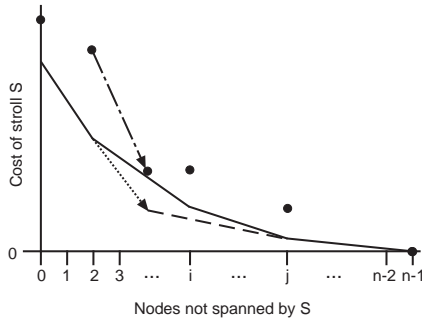


Figure 3: Pruning the anchoring strolls to achieve the $\beta\lambda_k^+$ -tail property.

the original point $(k, c(R_k))$. In Figure 3, this is represented by the dash-dotted line segment pointing from R_k to R_{k+z^*} . This line should be at least β times as steep as the dotted line (slope $-\beta\lambda_k^+$ vs. slope $-\lambda_k^+$).

- Extend the lower hull to the right of k at slope $-\lambda_k^+$. Notice that it will remain within a factor of beta of the new stroll, and will also still be a valid lower bound (because the next segment of the original lower hull lies strictly above it). Now, recompute the lower hull from $z+k$ to the right, and iterate this surgery process with the next corner point in the new lower hull. In Figure 3, recomputing the lower hull causes the dotted segment and the dashed segment to replace the two solid segments directly above them. Thus, i ceases to be a corner point in the new hull, so the next corner point we consider will be j .

After the last iteration of this surgery process, the resulting lower hull is an anchored β -lower envelope, and the corresponding corner point strolls are its anchors. ■

6 Derandomizing the 3.03-approximation algorithm

In this section, we show how to de-randomize the $\rho(\frac{1}{3}) \approx 3.03$ -approximation algorithm of Section 3, proving Theorem 3.3.

Deterministic 3.03-Approximation Algorithm Input: A finite metric space M on set V of n vertices.

1. Run Algorithm 1 on M , with $a = \frac{1}{3}$. Let $R_{p(1)}, R_{p(2)}, \dots, R_{p(k)}$ be the sequence of strolls it selects to concatenate.
2. Let c_i^{RT} be the actual latency of the RT traversal of $R_{p(i+1)}$ after traversing the sequence of strolls $R_{p(1)}, \dots, R_{p(i)}$. Similarly, let c_i^{TR} and c_i^{TT} be the latency for the TR and TT traversals respectively, includ-

ing the latency induced by backtracking along $R_{p(i)}$ to reach the first node in $R_{p(i+1)}$.

3. Build a graph CG_{RT} with nodes r_i and t_i for $i = 0, \dots, k$, edges $(r_i, t_{i+1}), (t_i, t_{i+1}), (t_i, r_{i+1})$ for $i = 0, \dots, (k-1)$, and edge costs $c_i^{RT}, c_i^{TT}, c_i^{TR}$ respectively.
4. Find the shortest path in CG_{RT} from node r_0 to the closer of t_k or r_k , then concatenate $R_{p(1)}, R_{p(2)}, \dots, R_{p(k)}$ according to which edges were used in the shortest path. If edge (r_{i-1}, t_i) is in the path, then we traverse $R_{p(i)}$ with an RT traversal; if edge (t_{i-1}, r_i) (respectively, (t_{i-1}, t_i)), then after traversing $R_{p(i-1)}$ we backtrack until we are at a node in $R_{p(i)}$, and then traverse $R_{p(i)}$ using the TR (respectively, TT) traversal.

The key to understanding this derandomization is that the actual latency of each RT , TR and TT traversal can be calculated, because when we traverse stroll $R_{p(i+1)}$, we know exactly which strolls we have already traversed (namely $R_{p(1)}, \dots, R_{p(i)}$), and hence we know exactly which nodes still have not been visited (namely $V - \cup_{j=1}^i V(R_{p(j)})$). Thus, the cost of any $r_0 \rightsquigarrow r_k$ or $r_0 \rightsquigarrow t_k$ path in CG_{RT} is precisely the latency of the corresponding tour. This is why it is necessary to fix the sequence of strolls first, because otherwise we cannot even compute the costs of the arcs in CG_{RT} .

7 Computing the lower stroll envelope for trees

This section deals exclusively with metric spaces that are specified as the shortest path metric on a weighted tree, rooted at node r . Lemma 2.11 applies to all graphs of constant treewidth, and we prove the general result in Section 8. However, the algorithm is *much* simpler when the graph is a tree, so we explain that result separately here.

We begin by explaining how to compute the optimal solution to the related prize-collecting Steiner tree (PCST) problem on trees. PCST is the same as PCS, except that the requirement on the set of edges we buy is that they form a tree, rather than a path. Johnson, Minkoff and Phillips [21] observed that the PCST problem on trees can be solved exactly in $O(n)$ time, although they phrased their algorithm as a “strong pruning” phase to be plugged into the Goemans-Williamson algorithm for PCST on general metric spaces[16]. For each node v in the tree, let $\text{ch}(v)$ denote its set of children, e_v denote its parent edge (which is null if $v = r$), and T_v denote the subtree rooted at v , consisting of v and all of its descendants. For each node v , we compute its *net worth* $NW(v)$ recursively as

$$NW(v) = \lambda - c(e_v) + \sum_{u \in \text{ch}(v)} NW(u)^+, \quad (7.24)$$

where the superscript $+$ denotes the positive part, i.e., $x^+ = \max(x, 0)$. In words, the net worth of a node v is the net benefit of buying edge e_v and claiming the λ prize at v , along with the best PCST solution for T_v . If $NW(v) > 0$, color the edge e_v green. The optimal solution is then the connected component of green edges that contains r .

For the duration of this section, we will think of a stroll as being a sequence of edges in the tree, not necessarily distinct. Alternatively, we can view it as a multiset of edges that admits an Eulerian path. In a tree, such a set of edges has a particularly simple structure: it is a path of edges with multiplicity one (which we call the *backbone path*), plus a collection of disjoint trees hanging off of this path, each with multiplicity two.

To compute the best PCS, we need to compute two types of net worth for each node: $NW_P(v)$ is the net worth of the node, given that the tail of the stroll lies in T_v (equivalently, the backbone path includes v), while $NW_T(v)$ is the net worth of the node, given that the tail does not lie in T_v . We can compute these recursively as:

$$NW_T(v) = \lambda - 2c(e_v) + \sum_{u \in \text{ch}(v)} NW_T(u)^+ \quad (7.25)$$

$$NW_P(v) = \lambda - c(e_v) + \sum_{u \in \text{ch}(v)} NW_T(u)^+ + \max_{w \in \text{ch}(v)} (NW_P(w)^+ - NW_T(w)^+) \quad (7.26)$$

For $NW_T(v)$, the path of multiplicity 1 does not run through v , so two copies of e_v must be purchased, whereas for $NW_P(v)$, only one copy need be purchased. Also, the backbone path can run through at most one of v 's children, and then only if it runs through v .

To extract the solution, we first color every edge e_v green for which $NW_T(v) \geq 0$. Then we perform an iterative process in which we color some edges red. First we evaluate whether any of the edges coming out of r has $NW_P(v) \geq 0$. If so, we let $w \in \text{ch}(r)$ be the $\arg \max$ from (7.26), and color e_w red. We then iterate this process, starting at node w . In this way, we identify a path of red edges. The edges of our final solution, will be those in the connected component of r with respect to the union of the red and green edges. The red edges will form the backbone path of our stroll, and the green edges will form the doubled trees. This algorithm gives the following theorem.

THEOREM 7.1. *The PCS problem on trees can be solved in $O(n)$ time, via dynamic programming.*

One can show that the $\arg \max$ and the positive parts in (7.25) and (7.26) can change at no more than $O(n)$ breakpoints as λ sweeps from 0 to ∞ , and the next breakpoint can always be computed in $O(n)$ time. This implies the following theorem.

THEOREM 7.2. *For trees, the lower stroll envelope and corresponding corner point strolls can be computed in time $O(n^2)$.*

8 Shortest path metrics on graphs of constant treewidth

In this section, we prove the following two Theorems, which subsume Lemma 2.11.

THEOREM 8.1. *Given a graph $G = (V, E_0)$ with constant treewidth and $\lambda \geq 0$, we can compute $S(\lambda)$, the optimal PCS with penalties λ , in $O(n)$ time.*

Sweeping λ from 0 to ∞ as in Theorem 7.2 gives the following Theorem.

THEOREM 8.2. *Given a graph $G = (V, E_0)$ with constant treewidth, the lower stroll envelope and corresponding corner point strolls can be computed in time $O(n^2)$.*

The proof of Theorem 8.1 will involve a hairy dynamic program. To define it, we must first review some definitions regarding treewidth. These are taken directly from Bodlaender [9].

DEFINITION 8.3. *A tree decomposition of a graph $G = (V, E)$ is a pair (T, \mathcal{X}) with $T = (J, F)$ a tree and $\mathcal{X} = \{X_j | j \in J\}$ a family of subsets of V , such that:*

- $\bigcup_j X_j = V$
- For all edges $(u, v) \in E$, $\exists j \in J$ such that $v \in X_j$ and $w \in X_j$.
- If X_j and X_k both contain a vertex $v \in V$, then $v \in X_i$ for all $i \in J$ on the (unique) path from j to k in T . That is, the nodes associated with vertex v form a connected subset of T .

In this section we will refer to elements of V as nodes and elements of J as supernodes.

DEFINITION 8.4. *The width of a tree decomposition $((J, F), \{X_j | j \in J\})$ is $\max_{j \in J} |X_j|$. The treewidth of a graph G is the minimum width over all tree decompositions of G .*

THEOREM 8.5. (BODLAENDER [8]) *For a graph G and a constant k , there exists a linear time algorithm that outputs a tree-decomposition of G with treewidth at most k , if one exists.*

To limit the number of recursive cases in our dynamic program for PCS we use an equivalent definition of tree-decomposition that has a more restricted structure.

DEFINITION 8.6. *A nice tree-decomposition is a tree-decomposition $((J, F), \{X_j | j \in J\})$, such that each supernode $i \in J$ has at most two children and*

- For supernodes j with no children, $|X_j| = 1$.
- For supernodes j with only one child k , either $X_j = X_k + \{v\}, v \notin X_k$ (j is an add supernode) or $X_j = X_k - \{v\}, v \in X_k$ (j is a drop supernode).
- For supernodes j with two children, k and l , $X_j = X_k = X_l$ (j is a join supernode).

THEOREM 8.7. (BODLAENDER [9]) *Given a graph with root r and a tree-decomposition T of width w , we can find a nice tree-decomposition in linear time with size linear in T , width w , and root i such that $X_i = \{r\}$.*

We now introduce some notation necessary to describe our dynamic program. Let D_j denote the set of nodes contained in the supernodes of the sub-tree of j but not j :

$$D_j = \left(\bigcup_{k \in \text{desc}(j)} X_k \right) - X_j,$$

where $\text{desc}(j)$ denotes the descendants of supernode j . If v is a node and E an edge set, let $d_v(E)$ denote the number of edges in E incident to v .

We observe that a set of edges is a tour if the graph induced on those edges is connected and Eulerian (every node has even degree). Similarly, a set of edges is a stroll starting at r if the induced graph is connected, r has odd degree, and all other nodes except one has even degree. Given this observation, we can view a solution to the PCS simply as a set of edges with these properties.

Now, we want to build a PCS using sub-solutions defined by our tree-decomposition, and we will view these sub-solutions also as a set of edges with certain properties.

We say a set of edges E is a *sub-solution* at j if:

1. $\forall (u, v) \in E$ $u \in D_j$ or $v \in D_j$ or both,
2. $d_u(E)$ is odd for at most one node $u \in D_j$,
3. $\forall (u, v) \in E$ there is a path from u to $x \in X_j$.

The first item defines what we mean by a sub-solution at j , as it tells us which subset of edges to consider. The second and third items ensure that the solution will extend to either a stroll or a tree.

Let $DISCONN(j, E)$ denote the set of nodes $u \in D_j$ for which there exists no path from u to any $v \in X_j$ using only edges in E . Now let

$$TC(j, E) = c(E) + \lambda |DISCONN(j, E)|,$$

that is, $TC(j, E)$ is the total cost (edge plus penalty) of a sub-solution E at supernode j . Observe that the minimum cost sub-solution of the root supernode is the optimal prize-collecting stroll.

We also define an interface of a supernode j that gives a description of a sub-solution at j in terms of the way the sub-solution affects the nodes in X_j . Specifically, an *interface* of a supernode j , denoted $I(j)$ is a 3-tuple $(\{P^i\}, D, s)$ where $\{P^i\} \subseteq 2^{X_j}$ is a partition of the nodes in X_j , $D : X_j \rightarrow \{0, 1, -1\}$ is a function of the nodes of X_j , and $s = \{0, 1\}$ is a bit. Because $|X_j|$ is less than some constant for all j , the number of possible interfaces for each j is also bounded by a constant.

We say a set of edges E is consistent with interface $I(j)$ if:

1. E is a sub-solution of j ,
2. $\exists i : u, v \in P^i \Leftrightarrow \exists$ a path in E from u to v ,
3. $\forall v \in X_j, d_v(E) = 0 \Leftrightarrow D(v) = -1$, otherwise, $d_v(E) \bmod 2 \equiv D(v)$,
4. if $s = 0$ then $\forall u \in D_j, d_u(E)$ is even.

The second item requires that the partition $\{P^i\}$ specifies which nodes of X_j are connected through paths in D_j . The third item requires that the function D gives the parity of the degree of any node in X_j (or indicates no adjacent nodes in which case we have -1). And the fourth item requires that the bit s is an indicator that all other nodes in D_j have even degree ($s = 0$) and thus our sub-solution can extend to a tour, or one node in D_j has odd degree ($s = 1$), and thus the sub-solution must extend to a stroll. Note that any sub-solution at j is consistent with exactly one interface of j .

We now describe the dynamic program. The dynamic program will use a nice tree-decomposition of our graph G with root R , $X_R = \{r\}$. Let $E_{I(j)}^*$ denote a set of edges consistent with $I(j)$ with minimal cost (there may be no such edge set). In our dynamic program we have a state for every supernode j , and for each state we will store $E_{I(j)}^*$ for each interface $I(j)$ if it exists.

Observe that any set of edges consistent with the root supernode R , $X_R = \{r\}$, of the tree-decomposition and interface $I(R) = (\{\{r\}\}, D(r) = 1, 1)$ is a stroll. And, $E_{I(R)}^*$ minimizes exactly objective of the PCS, and thus is the optimal PCS. So when we have found $E_{I(j)}^*$ for all interfaces we have found the optimal PCS.

The base case of our recurrence is the set of leaf supernodes. Every leaf node l has $X_l = \{v\}, D_l = \emptyset$, so for every interface $I(l)$, there is either no set of edges consistent with $I(l)$ or the empty set is consistent, and thus we can compute these sub-solutions easily. For our recurrence, we consider all possible ways to raise the solutions for the children to a parent solution as follows:

JOIN: Parent j , children k, l . Let $\mathcal{E} = \bigcup_{I(k), I(l)} E_{I(k)}^* \cup E_{I(l)}^*$. We show that any optimal sub-solution at node j is an element of \mathcal{E} . Consider $E_{I(j)}^*$

for some $I(j)$. Any edge $e \in E_{I(j)}^*$ can be in a sub-solution of exactly one of k or l by definition of a sub-solution and a tree decomposition. This induces a partition $E_{I(j)}^*$ into E_k, E_l where E_k and E_l are sub-solutions at E_k and E_l respectively. E_k and E_l are each consistent with exactly one interface of k : $I^*(k)$ and l : $I^*(l)$, respectively. Now, we claim that $E_k = E_{I^*(k)}^*$ and $E_l = E_{I^*(l)}^*$. The cost of $E_{I(j)}^*$ is simply the sum of the costs of E_k and E_l , and $E_{I^*(k)}^* \cup E_{I^*(l)}^*$ is consistent with $I(j)$ because the interface of $E_{I^*(k)}^* \cup E_{I^*(l)}^*$ is determined completely by the interfaces of $I^*(k)$ and $I^*(l)$ (this can be easily verified). Now, to determine the optimal sub-solutions at j , simply find the minimum cost sub-solution for each interface of j from \mathcal{E} .

DROP: Parent j , child k , node w is added to j . The argument is similar to the JOIN argument but even simpler, so we omit it. Optimal solutions for j are exactly the optimal solutions for k .

ADD: Parent j , child k , node w is dropped from j . Let $\mathcal{E} = \bigcup_{I(k), E \subseteq \{(u,w) | u \in X_j\}} E_{I(k)}^* \cup E$. We show that any optimal sub-solution at node j is an element of \mathcal{E} . Consider $E_{I(j)}^*$ for some $I(j)$. Any edge $e \in E_{I(j)}^*$ can be in the sub-solution of k or is adjacent to w by definition of a sub-solution and a tree decomposition. So, let E_k be the subset of edges of $E_{I(j)}^*$ not adjacent to w and let E_w be the set of edges in $E_{I(j)}^*$ adjacent to w . E_k is consistent with some interface $I^*(k)$. Now, we claim that E_k is the optimal solution for $I^*(k)$. This is because the union of any sub-solutions consistent with $I^*(k)$ and E_w gives a sub-solution consistent with $I(j)$ as it is easy to verify that the interface of $E_k \cup E_w$ is determined completely by the interface of E_k and the edge set E_w . Furthermore, the cost of $E_{I(j)}^*$ is $c(E_w) + TC(k, E_k)$. Again, to determine the optimal sub-solutions at j , simply find the minimum cost sub-solution for each interface of j from \mathcal{E} .

We have a constant number of interfaces for each node (though exponential in our constant treewidth) and each set \mathcal{E} is also constant, so to raise a sub-solution from a child to a parent takes constant time. Our tree decomposition has $O(n)$ nodes, and thus the dynamic program runs in $O(n)$ time, albeit with a huge hidden constant.

9 Acknowledgments

We thank David Applegate, Bobby Kleinberg and Mike Todd for helpful conversations regarding infinite dimensional linear programming.

References

[1] Edward J. Anderson and Peter Nash. *Linear Programming in Infinite Dimensional Spaces*. Wiley, 1987.
 [2] Aaron Archer and Shankar Krishnan. Importance sampling via load-balanced facility location. In *Proceedings of the*

13th International Conference on Integer Programming and Combinatorial Optimization, pages 316–330, 2008.

- [3] Aaron Archer, Asaf Levin, and David P. Williamson. A faster, better approximation algorithm for the minimum latency problem. *SIAM J. Comput.*, 37(5):1472–1498, 2008.
 [4] Sanjeev Arora and George Karakostas. Approximation schemes for minimum latency problems. *SIAM Journal on Computing*, 32(5):1317–1337, 2003.
 [5] Giorgio Ausiello, Stefano Leonardi, and Alberto Marchetti-Spaccamela. On salesmen, repairmen, spiders and other traveling agents. In *Proceedings of the Italian Conference on Algorithms and Complexity*, pages 1–16, 2000.
 [6] Nikhil Bansal, Lisa Fleischer, Tracy Kimbrel, Mohammad Mahdian, Baruch Schieber, and Maxim Sviridenko. Further improvements in competitive guarantees for QoS buffering. In *Proceedings of the 31st International Colloquium on Automata, Languages, and Programming (ICALP)*, pages 196–207, 2004.
 [7] Avrim Blum, Prasad Chalasani, Don Coppersmith, Bill Pulleyblank, Prabhakar Raghavan, and Madhu Sudan. The minimum latency problem. In *Proceedings of the 26th Annual ACM Symposium on Theory of Computing*, pages 163–171, 1994.
 [8] Hans L. Bodlaender. A linear-time algorithm for finding tree-decompositions of small treewidth. In *Proceedings of the 25th Annual ACM Symposium on Theory of Computing*, pages 226–234, 1993.
 [9] Hans L. Bodlaender. Treewidth: Algorithmic techniques and results. In *Proceedings of the 22nd International Symposium on Mathematical Foundations of Computer Science*, pages 19–36. Lecture Notes in Computer Science 1295, Springer, 1997.
 [10] Kamalika Chaudhuri, Brighten Godfrey, Satish Rao, and Kunal Talwar. Paths, trees, and minimum latency tours. In *Proceedings of the 44th Annual IEEE Symposium on the Foundations of Computer Science*, pages 36–45, 2003.
 [11] Fabian A. Chudak, Tim Roughgarden, and David P. Williamson. Approximate k -MSTs and k -Steiner trees via the primal-dual method and Lagrangean relaxation. *Mathematical Programming*, 100:411–421, 2004.
 [12] Vasek Chvatal. A greedy heuristic for the set covering problem. *Mathematics of Operations Research*, 4:233–235, 1979.
 [13] Naveen Garg. A 3-approximation for the minimum tree spanning k vertices. In *Proceedings of the 37th Annual Symposium on Foundations of Computer Science*, pages 302–309, 1996.
 [14] Michel Goemans and Jon Kleinberg. An improved approximation ratio for the minimum latency problem. *Mathematical Programming*, 82:111–124, 1998.
 [15] Michel X. Goemans and Jon M. Kleinberg. An improved approximation ratio for the minimum latency problem. In *Proceedings of the 7th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 152–158, 1996.
 [16] Michel X. Goemans and David P. Williamson. A general approximation technique for constrained forest problems. *SIAM Journal on Computing*, 24:296–317, 1995.
 [17] Martin Grötschel, László Lovász, and Alexander Schri-

- jver. *Geometric Algorithms and Combinatorial Optimization*. Springer, 1993.
- [18] Nicole Immorlica, Mohammad Mahdian, and Vahab S. Mirrokni. Cycle cover with short cycles. In *Proceedings of the 22nd Annual Symposium on Theoretical Aspects of Computer Science (STACS)*, pages 641–653, 2005.
 - [19] Kamal Jain, Mohammad Mahdian, and Mohammad R. Salavatipour. Packing Steiner trees. In *Proceedings of the 14th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 266–274, 2003.
 - [20] Kamal Jain and Vijay V. Vazirani. Approximation algorithms for metric facility location and k -median problems using the primal-dual schema and Lagrangian relaxation. *Journal of the ACM*, 48(2):274–296, 2001.
 - [21] David S. Johnson, Maria Minkoff, and Steven Phillips. The prize collecting Steiner tree problem: theory and practice. In *Proceedings of the 11th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 760–769, 2000.
 - [22] Jon Kleinberg. Personal communication, 2001.
 - [23] Elias Koutsoupias, Christos H. Papadimitriou, and Mihalis Yannakakis. Searching a fixed graph. In *Proceedings of the 23rd International Colloquium on Automata, Languages, and Programming*, pages 280–289, 1996.
 - [24] Mohammad Mahdian. *Facility Location and the Analysis of Algorithms through Factor-Revealing Programs*. PhD thesis, MIT, Cambridge, MA, June 2004.
 - [25] Mohammad Mahdian, Yinyu Ye, and Jiawei Zhang. Improved approximation algorithms for metric facility location problems. In *Proceedings of the 5th International Workshop on Approximation Algorithms for Combinatorial Optimization Problems (APPROX)*, pages 229–242, 2002.
 - [26] Nimrod Megiddo. Combinatorial optimization with rational objective functions. *Mathematics of Operations Research*, 4:414–424, 1979.
 - [27] Christos H. Papadimitriou and Mihalis Yannakakis. The traveling salesman problem with distances one and two. *Mathematics of Operations Research*, 18:1–11, 1993.
 - [28] Walter Rudin. *Real and Complex Analysis*. McGraw-Hill, 1974.
 - [29] Sartaj Sahni and Teofilo Gonzalez. P-complete approximation problems. *Journal of the ACM*, 23:555–565, 1976.
 - [30] Alexander Schrijver. *Theory of Linear and Integer Programming*. Wiley, 1998.
 - [31] René Sitters. The minimum latency problem is NP-hard for weighted trees. In *Proceedings of the 9th Conference on Integer Programming and Combinatorial Optimization*, pages 230–239, 2002.