Linear-Programming based Approximation Algorithms for Multi-Vehicle Minimum Latency Problems

(Extended Abstract)

Ian Post* Chaitanya Swamy*

Abstract

We consider various *multi-vehicle versions of the minimum latency problem*. There is a fleet of k vehicles located at one or more depot nodes, and we seek a collection of routes for these vehicles that visit all nodes so as to minimize the total latency incurred, which is the sum of the client waiting times. We obtain an 8.497-approximation for the version where vehicles may be located at multiple depots and a 7.183-approximation for the version where all vehicles are located at the same depot, both of which are the first improvements on this problem in a decade. Perhaps more significantly, our algorithms exploit various LP-relaxations for minimum-latency problems. We show how to effectively leverage two classes of LPs—*configuration LPs* and *bidirected LP-relaxations*—that are often believed to be quite powerful but have only sporadically been effectively leveraged for network-design and vehicle-routing problems. This gives the first concrete evidence of the effectiveness of LP-relaxations for this class of problems.

The 8.497-approximation the multiple-depot version is obtained by rounding a near-optimal solution to an underlying configuration LP for the problem. The 7.183-approximation can be obtained both via rounding a bidirected LP for the single-depot problem or via more combinatorial means. The latter approach uses a bidirected LP to obtain the following key result that is of independent interest: for any k, we can efficiently compute a rooted tree that is at least as good, with respect to the prize-collecting objective (i.e., edge cost + number of uncovered nodes) as the best collection of k rooted paths. This substantially generalizes a result of Chaudhuri et al. [11] for $k = 1$, yet our proof is significantly simpler. Our algorithms are versatile and extend easily to handle various extensions involving: (i) weighted sum of latencies, (ii) constraints specifying which depots may serve which nodes, (iii) node service times.

Finally, we propose a configuration LP that sheds further light on the power of LP-relaxations for minimum-latency problems. We prove that the integrality gap of this LP is at most 3.592, even for the multi-depot problem, both via an efficient rounding procedure, and by showing that it is at least as powerful as a stroll-based lower bound that is oft-used for minimum-latency problems; the latter result implies an integrality gap of at most 3.03 when $k = 1$. Although, we do not know how to solve this LP in general, it can be solved (near-optimally) when $k = 1$, and this yields an LP-relative 3.592-approximation for the single-vehicle problem, matching (essentially) the current-best approximation ratio for this problem.

1 Introduction

Vehicle-routing problems constitute a broad class of combinatorial-optimization problems that find a wide range of applications and have been widely studied in the Operations Research and Computer Science communities (see, e.g., [27]). A fundamental vehicle-routing problem is the *minimum latency problem* (MLP), variously known as the *traveling repairman problem* or the *delivery man problem* [1, 22, 15, 8], wherein, taking a client-oriented perspective, we seek a route starting at a given root node that visits all

*ian@ianpost.org, cswamy@uwaterloo.ca. Dept. of Combinatorics and Optimization, Univ. Waterloo, Waterloo, ON N2L 3G1. Supported in part by NSERC grant 327620-09. The second author is also supported by an NSERC Discovery Accelerator Supplement Award and an Ontario Early Researcher Award.

client nodes and minimizes the total client waiting time. We consider various multi-vehicle versions of the minimum latency problem (MLP). In these problems, there is a fleet of k vehicles located at one or more depot nodes, and we seek a collection of routes for these vehicles that together visit all the client nodes so as to minimize the total latency incurred, which is the sum of the client waiting times.

Formally, we consider the *multi-depot k-vehicle minimum latency problem* (multi-depot k-MLP), which is defined as follows. We are given a complete undirected graph $G = (V, E)$ on n nodes, with metric edge costs $\{c_e\}$, and k not necessarily distinct root/depot nodes $r_1, \ldots, r_k \in V$. A feasible solution consists of k paths $P_1, \ldots, P_k$, where each path P_i starts at root r_i, such that the P_is cover all the nodes. The waiting time or *latency* of a node v that is visited by path P_i, is the distance from r_i to v along P_i, and is denoted by $c_{P_i}(v)$. The goal is to minimize the *total latency* $\sum_{i=1}^{k} \sum_{v \in P_i : v \neq r_i} c_{P_i}(v)$ incurred.[1] (Due to metric costs, one may assume that any two P_is are node disjoint, unless they share a common root, which is then the only shared node.) We refer to the special case where all depots are identical, i.e., $r_1 = r_2 = \ldots = r_k$, as *single-depot* k-MLP, which we abbreviate simply to k-MLP.

In addition to self-evident applications in logistics, the problem of finding an optimal routing as modeled by multi-depot k-MLP can also be motivated from the perspective of searching a graph (e.g., the web graph) for a hidden treasure [8, 20, 5]; if the treasure is placed at a random node of the graph then multi-depot k-MLP captures the problem of minimizing the expected search time using k search agents (e.g., web crawlers). Even 1-MLP is known to be *MAXSNP*-hard for general metrics [8, 23] and *NP*-hard for tree metrics [26], so we focus on approximation algorithms.

Our results and techniques. We obtain approximation guarantees of 8.497 for multi-depot k-MLP (Theorem 5.1) and 7.183 for (single-depot) k-MLP (Theorem 6.6), which are the first improvements on the respective problems in over a decade. The previous best approximation ratios for the multi- and single- depot problems were 12 (by combining [12, 11]) and 8.497 (by combining [13, 11]; see also [14]) respectively.

Perhaps more significantly, as we elaborate below, our algorithms exploit various linear-programming (LP) relaxations, including various *configuration-style LP relaxations* as well as a *bidirected LP-relaxation*. This is noteworthy for two reasons. First, it gives the *first concrete evidence* of the effectiveness of LP-relaxations for minimum latency problems. Second, we show how to effectively leverage two classes of LPs—bidirected LPs and configuration LPs—that are often believed to be quite powerful but have only sporadically been effectively leveraged for network-design and vehicle-routing problems. Previously, Chakrabarty and Swamy [10] had proposed some LP-relaxations (including a configuration LP) for minimum-latency problems but they could not improve upon the current-best approximation guarantees for these problems via these LPs. Our LPs are inspired by their formulations, and coincide with their LPs in some cases, but are subtly stronger, and, importantly (as noted above), our guarantees do indeed improve the state-of-the-art for these problems and testify to the effectiveness of LP-based methods.

Our algorithms are versatile and extend easily to handle various extensions involving weighted sum of node latencies, node-depot service constraints, and node service times (Section 7).

Finally, we propose a configuration LP that sheds further light on the power of LP-relaxations for minimum-latency problems and why they merit further investigation. We prove that this LP has integrality gap at most $\mu^* < 3.5912$ for (the general setting of) multi-depot k-MLP (see Theorem 4.2), both by devising an efficient rounding procedure, and by showing that this LP is at least as strong as a combinatorial stroll-based bound that is frequently used for minimum-latency problems, especially when $k = 1$ for which this leads to the current-best approximation ratio. The latter result implies an integrality gap of at most 3.03 when $k = 1$ (due to the analysis of the ℓ-stroll lower bound in [2]). We do not know how to solve this LP efficiently in general, but we can efficiently obtain a $(1 + \epsilon)$-approximate solution when $k = 1$, and thereby efficiently

[1] Multi-depot k-MLP is often stated in terms of finding k tours starting at $r_1, \ldots, r_k$; since the last edge on a tour does not contribute to the latency of any node, the two formulations are equivalent. We find the path-formulation to be more convenient.

obtain an LP-relative $(\mu^* + \varepsilon)$-approximation, which essentially matches the current-best approximation ratio for single-vehicle (i.e., $k = 1$) MLP.

We now sketch the main ideas underlying our algorithms and analyses. Our 8.497-approximation for multi-depot k-MLP (Section 5) leverages a natural configuration LP (LP$_{\mathcal{P}}$), where we define a configuration for time t and vehicle i to be an r_i-rooted path of length at most t. Using known results on orienteering [11] and the arguments in [10], one can compute a fractional "solution" of cost at most $OPT_{\mathcal{P}}$, where (roughly speaking) the solution computed specifies for every t and i, a distribution over r_i-rooted *trees* (instead of paths) of length roughly t, such that this ensemble covers nodes to the appropriate extents (Lemma 4.1). The rounding algorithm is then simple: we consider time points of geometrically increasing value, we sample a tree for each time point t and vehicle i from the distribution for t, i, convert this tree to a tour, and finally, for each vehicle i, we concatenate the various tours obtained for i to obtain i's route. Compared to the combinatorial algorithm in [12], we gain significant savings from the fact that the LP solution readily yields, for each time t, a random k-tuple of trees whose expected coverage can be related to the coverage of the LP solution. In contrast, [12] devise an algorithm for the cost version of their variant of max-coverage to obtain a k-tuple of trees with the desired coverage and lose various factors in the process.

The 7.183-approximation algorithm for single-depot k-MLP (Section 6) relies crucially on *bidirected LPs*. We obtain this guarantee both by rounding a compact bidirected LP-relaxation for the problem and via more combinatorial arguments where we utilize bidirected LPs to furnish a key ingredient of the algorithm. The bidirected LP-relaxation (LP3) (which is a relaxation for multi-depot k-MLP) works with the digraph obtained by bidirecting the edges of the input (complete) graph. The LP specifies the extent to which nodes are covered by each vehicle at each time, and the extent $z^i_{a,t}$ to which each arc a has been traversed by vehicle i's route up to time t. We impose suitable node-degree and node-connectivity constraints on the $z^i_{a,t}$s, and observe that, for each time t, one can then use (a polytime version of) the arborescence-packing results of Bang-Jensen et al. [6] (Theorem 3.1) to decompose $\{\sum_i z^i_{a,t}\}_a$ into a distribution of r-rooted trees with expected length at most kt and covering at least as many nodes as the LP does by time t. We convert each tree in the support into a k-tuple of tours (of length at most $4t$) and stitch together these tours using a concatenation-graph argument similar to the one in [18] (losing a $\frac{\mu^*}{2}$-factor), which also shows that the fact that we have a distribution of trees for each t instead of a single trees does not cause any problems. Theorem 3.1 is precisely what leads to our improvement over [14]: we *match* the coverage of an optimal LP-solution at each step (incurring a certain blow-up in cost), whereas [14] sequentially find k tours, causing them to lag behind in coverage and incur a corresponding loss in approximation.

The combinatorial arguments rely on the following result that is of independent interest. We show that one can efficiently compute an r-rooted tree that is at least as good, with respect to a prize-collecting objective that incorporates both the edge cost and the number of nodes covered, as the best collection of (any number of, and perhaps non-simple) r-rooted paths (Theorem 3.2). We obtain this by formulating a bidirected LP for the prize-collecting problem of finding the desired collection of paths, and rounding it using a polytime version (that we prove) of the arborescence-packing results of [6] for weighted digraphs. Theorem 3.2 also implies that for every ℓ, one can efficiently compute an r-rooted tree, or a distribution over two r-rooted trees, which we call a *bipoint tree*, that, in expectation, spans at least ℓ nodes and has cost at most the minimum total cost of a collection of r-rooted paths spanning ℓ nodes (Corollary 3.3). Again, this is where we improve over [14] since we match the coverage of an optimal (integer) solution at each step. We compute these objects for all values of ℓ, convert each constituent tree into k tours, and stitch them together as before.

Theorem 3.2 relating prize-collecting trees and path-collections substantially generalizes a result of Chaudhuri et al. [11], who prove an analogous result for the special case where one compares (the computed tree) against the best *single path*. Whereas this suffices for *single-vehicle* MLP (and allowed [11] to improve the approximation for single-vehicle MLP), it does not help in multiple-vehicle settings. (This is because to lower bound the ℓ-th smallest latency incurred by the optimal solution, one needs to consider a collection of

k *paths* that together cover ℓ nodes.) Notably, our proof of our more general result is significantly simpler and cleaner (and different) than the one in [11]. We remark that the approach in [11] (for $k = 1$), where one "guesses" the endpoint of an optimum path, is computationally infeasible for large k.

Related work. Although single-vehicle MLP (which we refer to simply as MLP) has attracted much attention in the Computer Science and Operations Research communities, there is little prior work on multi-vehicle versions of MLP, especially from the perspective of approximation algorithms. Chekuri and Kumar [12], and Fakcharoenphol et al. [13] seem to be the first ones to consider multi- and single- depot k-MLP respectively; they obtained approximation ratios of 12β and 8.497β for these problems respectively, where β is the approximation ratio of the ℓ-MST problem.[2] Subsequently, a result of Chaudhuri et al. [11] relating prize-collecting trees and paths provided a general tool that allows one eliminate the β term in the above approximation ratios. Recently Sitters [25] obtained a PTAS for MLP on trees, the Euclidean plane, and planar graphs, and mentions that the underlying techniques extend to yield a PTAS on these graphs for (*single-depot*) k-MLP for any *constant* k. We are not aware of any other work on k-MLP.

We now discuss some relevant work on MLP (i.e., $k = 1$). MLP (and hence k-MLP) is known to be hard to approximate to better than some constant factor [8, 23], and *NP*-hard even on trees [26]. While much work in the Operations Research literature has focused on exactly solving MLP (in exponential time) [21, 24, 15, 7], Blum et al. [8] devised the first constant-factor approximation for MLP via the process of finding tours of suitable lengths and/or node coverages and concatenating them. They obtained a $\min\{144, 8\beta\}$-approximation. Subsequently, [18] refined the concatenation procedure in [8] and proposed the device of a concatenation graph to analyze this process, which yielded an improved $\mu^*\beta$-approximation, where $\mu^* < 3.5912$ is the solution to $\mu \ln \mu = 1 + \mu$. The procedure of stitching together tours and its analysis via the concatenation graph have since become standard tools in the study of minimum-latency problems. Archer et al. [3] showed that one can replace the ℓ-MST-subroutine in the algorithm of [18] by a so-called *Lagrangian-multiplier preserving* (LMP) β'-approximation algorithm for the related *prize-collecting Steiner tree* (PCST) problem, and thereby achieve a $2\mu^*$-approximation using the LMP 2-approximation for PCST [19]. The current-best approximation for MLP is due to Chaudhuri et al. [11] who showed that the factors β and β' above can be eliminated, leading to a μ^*-approximation, by noting that: (i) the lower bound $\sum_{\ell=1}^{n}$(optimal value of ℓ-MST) used in all previous works starting with [8] can be strengthened to the *ℓ-stroll lower bound* by replacing the summand with the optimal cost of a rooted *path* covering ℓ nodes; and (ii) one can adapt the arguments in [19, 16] to obtain a prize-collecting tree of cost no more than that of an optimal prize-collecting tree. As noted earlier, (ii) is a rather special case of our Theorem 3.2.

Chakrabarty and Swamy [10] proposed some LP relaxations for minimum-latency problems and suggested that their LPs may lead to improvements for these problems. This was the inspiration for our work. Our LPs are subtly different, but our work lends credence to the idea that LP-relaxations for minimum-latency problems can lead to improved guarantees for these problems.

Improved guarantees are known for MLP in various special cases. Arora and Karakostas [4] give a quasi-PTAS for trees, and Euclidean metrics in any finite dimension. Sitters [25] recently improved these to a PTAS for trees, the Euclidean plane, and planar graphs.

2 Preliminaries

Recall that in the *multi-depot k-vehicle minimum latency problem* (multi-depot k-MLP), we have a complete undirected graph $G = (V, E)$ on n nodes, metric edge costs $\{c_e\}$, and a set $R = \{r_1, \ldots, r_k\}$ of k root/depot

[2] The 12β-approximation in [12] is a consequence of the algorithm they develop for the cost-version of the max-coverage variant that they consider. Although the cardinality-version of their problem now admits a better approximation ratio since it can be cast as a submodular-function maximization problem subject to a matroid constraint [9], this improvement does not apply to the cost-version, which gives rise to multiple knapsack constraints.

nodes. The goal is to find k paths $P_1, \ldots, P_k$, where each path P_i starts at r_i, so that $\bigcup_{i=1}^{k} V(P_i) = V$, and the total latency $\sum_{i=1}^{k} \sum_{v \in P_i : v \neq r_i} c_{P_i}(v)$ incurred is minimized. We call the special case where $r_1 = \ldots = r_k$ *single-depot* k-MLP and abbreviate this to k-MLP. We sometimes refer to non-root nodes as clients. We may assume that the c_es are integers, and $c_{r_i v} \geq 1$ for every non-depot node v and every root r_i.

Algorithms for minimum-latency problems frequently use the idea of concatenating tours to build a solution, and our algorithms also follow this general template. A *concatenation graph* [18] is a convenient means of representing this concatenation process. The concatenation graph corresponding to a sequence $C_1 = 0, \ldots, C_n$ of nonnegative numbers (such as the lengths of tours spanning $1, 2, \ldots, n$ nodes) denoted $CG(C_1, \ldots, C_n)$, is a directed graph with n nodes, and an arc (i, j) of length $C_j\left(n - \frac{i+j}{2}\right)$ for all $i < j$. We collect below some useful facts about this graph. We say that C_ℓ is an *extreme point* of the sequence $(C_1, \ldots, C_n)$ if (ℓ, C_ℓ) is extreme-point of the convex hull of $\{(j, C_j) : j = 1, \ldots, n\}$.

Theorem 2.1 ([18, 3, 2]). *The shortest $1 \rightsquigarrow n$ path in $CG(C_1, \ldots, C_n)$ has length at most $\frac{\mu^*}{2} \sum_{\ell=1}^{n} C_\ell$, where $\mu^* < 3.5912$ is the solution to $\mu \ln \mu = \mu + 1$. Moreover, the shortest path only visits nodes corresponding to extreme points of $(C_1, \ldots, C_n)$.*

Given a point-set $S \subseteq \mathbb{R}_+^2$, define its *lower-envelope curve* $f : [\min_{(x,y)\in S} x, \max_{(x,y)\in S} x] \mapsto \mathbb{R}_+$ by $f(x) = \min\{y : y \in \text{conv}(S)\}$, where $\text{conv}(S)$ denotes the convex hull of S. Note that f is well defined since the minimum is taken over a closed, compact set.

Let f be the lower-envelope curve of $\{(j, C_j) : j = 1, \ldots, n\}$, where $C_1 = 0$. If C_ℓ is an extreme point of $(C_1, \ldots, C_n)$, we will often say that (ℓ, C_ℓ) is a "corner point" of f. Notice that the bound in Theorem 2.1 on the shortest-path length can be strengthened to $\frac{\mu^*}{2} \sum_{\ell=1}^{n} f(\ell)$. This is because the shortest path P_f in the concatenation graph $CG(f(1), \ldots, f(n))$ has length at most $\frac{\mu^*}{2} \sum_{\ell=1}^{n} f(\ell)$, and uses only extreme points of $\big(f(1), \ldots, f(n)\big)$, which in turn must be extreme points of $(C_1, \ldots, C_n)$ since f is the lower-envelope curve of $\{(j, C_j) : j = 1, \ldots, n\}$. Hence, P_f is also a valid path in $CG(C_1, \ldots, C_n)$, and its length in these two graphs is exactly the same. Corollary 2.2 shows that this bound can further be strengthened to $\frac{\mu^*}{2} \int_1^n f(X)dx$. This will be useful in Section 6. The proof is similar to the above argument and follows by discretizing f using finer and finer scales; we defer the proof to Appendix A.

Corollary 2.2. *The shortest $1 \rightsquigarrow n$ path in $CG(C_1, \ldots, C_n)$ has length at most $\frac{\mu^*}{2} \int_1^n f(x)dx$, where $f : [1, \ldots, n] \mapsto \mathbb{R}_+$ is the lower-envelope curve of $\{(j, C_j) : j = 1, \ldots, n\}$, and only visits nodes corresponding to extreme points of $(C_1, \ldots, C_n)$.*

The bottleneck-stroll lower bound. Our algorithms for single-depot k-MLP utilize a combinatorial stroll-based lower bound that we call the (k, ℓ)-*bottleneck-stroll lower bound* (that applies even to multi-depot k-MLP), denoted by BNSLB, which is obtained as follows. Given an instance $\big(G = (V, E), \{c_e\}, k, R = \{r_1, \ldots, r_k\}\big)$ of multi-depot k-MLP, in the (k, ℓ)-*bottleneck-stroll* problem, we seek k paths $P_1, \ldots, P_k$, where each P_i is rooted at r_i, that together cover at least ℓ nodes (that may include root nodes) so as to minimize $\max_i c(P_i)$. Let $\mathsf{BNS}(k, \ell)$ denote the cost of an optimal solution. It is easy to see that if t^*_ℓ is the ℓ-th smallest node latency incurred by an optimal solution, then $t^*_\ell \geq \mathsf{BNS}(k, \ell)$. We define $\mathsf{BNSLB} := \sum_{\ell=1}^{|V|} \mathsf{BNS}(k, \ell)$, which is clearly a lower bound on the optimum value of the multi-depot k-MLP instance.

To put this lower bound in perspective, we remark that a concatenation-graph argument dovetailing the one used for MLP in [18] shows that there is multi-depot-k-MLP solution of cost at most $\mu^* \cdot \mathsf{BNSLB}$ (see Theorem 2.3). For $k = 1$, BNSLB becomes the ℓ-stroll lower bound in [11], which yields the current-best approximation factor for MLP. Also, the analysis in [2] shows that there is an MLP-solution of cost at most $3.03 \cdot \mathsf{BNSLB}$. On the other hand, whereas such combinatorial stroll-based lower bounds have been frequently leveraged for minimum-latency problems, we provide some evidence in Section 8 that LPs may prove to be

even more powerful by describing a configuration LP whose optimal value is always at least BNSLB. We defer the proof of the following theorem to Appendix A.

Theorem 2.3. *(i) There is a solution to multi-depot k-MLP of cost at most $\mu^* \cdot$ BNSLB. (ii) There is a solution to MLP of cost at most $3.03 \cdot$ BNSLB.*

3 Arborescence packing and prize-collecting trees and paths

A key component of our algorithms for (single-depot) k-MLP is a polytime version of an arborescence-packing result of [6] that we prove for weighted digraphs (Theorem 3.1). We use this to obtain a result relating trees and paths that we utilize in our "combinatorial" algorithm for k-MLP, which we believe is of independent interest. Let r be a given root node. Throughout this section, when we say rooted tree or rooted path, we mean a tree or path rooted at r. We show that for any $\lambda > 0$, one can efficiently find a rooted tree T such that $c(T) + \lambda|V \setminus V(T)|$ is at most $\sum_{P\in\mathcal{C}} c(P) + \lambda|V \setminus \bigcup_{P\in\mathcal{C}} V(P)|$, where $\mathcal{C}$ is any collection of rooted paths (see Theorem 3.2). As noted earlier, this substantially generalizes a result in [11], yet our proof is simpler.

For a digraph D (possibly with parallel edges), we use $\lambda_D(x, y)$ to denote the number of $x \rightsquigarrow y$ edge-disjoint paths in D. Given a digraph D with nonnegative integer edge weights $\{w_e\}$, we define the quantities $|\delta^{\text{in}}(u)|$, $|\delta^{\text{out}}(u)|$ and $\lambda_D(x, y)$ for D to be the respective quantities for the unweighted (multi-)digraph obtained by replacing each edge e of D with w_e parallel edges (that is, $|\delta^{\text{in}}(u)| = \sum_{e=(\circ,u)} w_e$ etc.). Bang-Jensen at al. [6] proved an arborescence-packing result that in particular implies that if $D = (U + r, A)$ is a digraph with root $r \notin U$ such that $|\delta^{\text{in}}(u)| \geq |\delta^{\text{out}}(u)|$ for all $u \in U$, then, for any integer $k \geq 0$, one can find k edge-disjoint out-arborescences rooted at r such that every node $u \in U$ belongs to at least $\min\{k, \lambda_D(r, u)\}$ arborescences. For a weighted digraph, applying this result on the corresponding unweighted digraph yields a pseudopolynomial-time algorithm for finding the stated arborescence family. We prove the following *polytime* version of their result for weighted digraphs; the proof appears in Appendix B.

Theorem 3.1. *Let $D = (U + r, A)$ be a digraph with nonnegative integer edge weights $\{w_e\}$, where $r \notin U$ is a root node, such that $|\delta^{\text{in}}(u)| \geq |\delta^{\text{out}}(u)|$ for all $u \in U$. For any integer $K \geq 0$, one can find out-arborescences $F_1, \ldots, F_q$ rooted at r and integer weights $\gamma_1, \ldots, \gamma_q$ in polynomial time such that $\sum_{i=1}^{q} \gamma_i = K$, $\sum_{i:e\in F_i} \gamma_i \leq w_e$ for all $e \in A$, and $\sum_{i:u\in F_i} \gamma_i = \min\{K, \lambda_D(r, u)\}$ for all $u \in U$.*

We now apply Theorem 3.1 to prove our key result relating prize-collecting arborescences and prize-collecting paths. Let $G = (V, E)$ be a complete undirected graph with root $r \in V$, metric edge costs $\{c_e\}$, and nonnegative node penalties $\{\pi_v\}_{v\in V}$. We bidirect the edges to obtain a digraph $D = (V, A)$, setting the cost of both (u, v) and (v, u) to c_{uv}. Consider the following bidirected LP-relaxation for the problem of finding a collection $\mathcal{C}$ of rooted paths minimizing $\sum_{P\in\mathcal{C}} c(P) + \pi\big(V \setminus \bigcup_{P\in\mathcal{C}} V(P)\big)$. We use a to index edges in A, and v to index nodes in $V \setminus \{r\}$.

$$\min \quad \sum_a c_a x_a + \sum_v \pi_v z_v \qquad \text{s.t.} \qquad x\big(\delta^{\text{in}}(v)\big) \geq x\big(\delta^{\text{out}}(v)\big) \quad \forall v \in V \setminus \{r\} \tag{PC-LP}$$

$$x\big(\delta^{\text{in}}(S)\big) + z_v \geq 1 \qquad \forall S \subseteq V \setminus \{r\}, v \in S; \qquad x, z \geq 0.$$

Theorem 3.2. *(i) We can efficiently compute a rooted tree T such that $c(T) + \pi(V \setminus V(T)) \leq OPT_{\text{PC-LP}}$. (ii) Hence, for any $\lambda \geq 0$, we can find a tree T_λ such that $c(T_\lambda) + \lambda|V \setminus V(T_\lambda)| \leq \sum_{P\in\mathcal{C}} c(P) + \lambda|V \setminus \bigcup_{P\in\mathcal{C}} V(P)|$ for any collection $\mathcal{C}$ of rooted paths.*

Proof. Let (x, z) be an optimal solution to (PC-LP). Let K be such that Kx_a is an integer for all a; note that $\log K$ is polynomially bounded in the input size. Consider the digraph D with edge weights $\{Kx_a\}$. Let $(\gamma_1, F_1), \ldots, (\gamma_q, F_q)$ be the weighted arborescence family obtained by applying Theorem 3.1 to D with

the integer K. Then, we have: (a) $\sum_{i=1}^{q} \gamma_i = K$; (b) $\sum_{i=1}^{q} \gamma_i c(F_i) = \sum_a c_a \big(\sum_{i:a\in F_i} \gamma_i\big) \le K \sum_a c_a x_a$; and (c) $\sum_{i=1}^{q} \gamma_i \pi\big(V \setminus V(F_i)\big) = \sum_v \pi_v \big(\sum_{i:v\notin F_i} \gamma_i\big) \le K \sum_v \pi_v z_v$, where the last inequality follows since $\lambda_D(r,v) \ge K(1-z_v)$ for all $v \in V \setminus \{r\}$. Thus, if we take the arborescence F_i with minimum prize-collecting objective $c(F_i) + \pi(V \setminus V(F_i))$, which we will treat as a rooted tree T in G, we have that $c(T) + \pi(V \setminus V(T)) \le \sum_a c_a x_a + \sum_v \pi_v z_v$.

For part (ii), let T_λ be the tree obtained in part (i) with penalties $\pi_v = \lambda$ for all $v \in V$. Observe that any collection $\mathcal{C}$ of rooted paths yields a feasible solution to (PC-LP) of cost $\sum_{P\in\mathcal{C}} c(P) + \lambda |V \setminus \bigcup_{P\in\mathcal{C}} V(P)|$.

Notice that the above proof does not use the fact that the edge costs are symmetric or form a metric; so the theorem statement holds with *arbitrary* nonnegative edge costs $\{c_a\}_{a\in A}$. ■

Given Theorem 3.2 for the prize-collecting problem, one can use binary search on the parameter λ to obtain the following result for the partial-cover version; the proof appears in Appendix C. A rooted *bipoint tree* $T = (a, T_1, b, T_2)$, where $a, b \ge 0,\ a + b = 1$, is a convex combination $aT_1 + bT_2$ of two rooted trees T_1 and T_2. We extend a function f defined on trees to bipoint trees by setting $f(T) = af(T_1) + bf(T_2)$.

Corollary 3.3. *(i) Let $B \ge 0$, and O^* be the minimum cost of a collection of rooted paths spanning at least B nodes. We can efficiently compute a rooted tree or bipoint tree Q such that $c(Q) \le O^*$ and $|V(Q)| = B$. (ii) Let $\{w_v\}$ be nonnegative node penalties with $w_r = 0$. Let $C \ge 0$, and n^* be the maximum node weight of a collection of rooted paths of total cost at most C. We can efficiently compute a rooted tree or bipoint tree Q such that $c(Q) = C$ and $w(V(Q)) \ge n^*$.*

4 LP-relaxations for multi-depot k-MLP

Our LP-relaxations are time-indexed formulations inspired by the LPs in [10]. Let $\mathsf{LB} := \max_v \min_i c_{r_i v}$. Let $\mathsf{T} \le 2n\mathsf{LB}$ be an upper bound on the maximum latency of a node that can be certified by an efficiently-computable solution. Standard scaling and rounding can be used to ensure that $\mathsf{LB} = \mathrm{poly}\big(\frac{n}{\epsilon}\big)$ at the expense of a $(1+\epsilon)$-factor loss (see, e.g., [4]). So we assume in the sequel that T is polynomially bounded. In Section 7, we sketch an approach showing how to solve our time-indexed LPs without this assumption, which turns out to be useful for some of the extensions that we consider. In either case, this means that all our LP-based guarantees degrade by a $(1+\epsilon)$ multiplicative factor. Throughout, we use v to index the non-root nodes in $V \setminus R$, i to index the k vehicles, and t to index time units in $[\mathsf{T}] := \{1, 2, \ldots, \mathsf{T}\}$.

In Section 4.1, we describe two *configuration LPs* with exponentially many variables. The first LP, ($\mathrm{LP}_{\mathcal{P}}$), can be "solved" efficiently and leads to an 8.497-approximation for multi-depot k-MLP (Section 5). The second LP, ($\mathrm{LP2}_{\mathcal{P}}$), is a stronger LP that we do not know how to solve efficiently (except when $k = 1$), but whose integrality gap is much smaller (see Theorem 4.2). In Section 4.2, we describe a *bidirected LP-relaxation* with exponentially many cut constraints that one can separate over and hence solve the LP (assuming T is polynomially bounded). This LP is weaker than ($\mathrm{LP}_{\mathcal{P}}$), but we show in Section 6 that this leads to a 7.813-approximation algorithm for k-MLP and a μ^*-approximation algorithm for MLP (i.e., $k = 1$). Theorem 4.3 summarizes the relationship between the various LPs and the guarantees we obtain relative to these via the algorithms described in the following sections.

4.1 Configuration LPs

The idea behind a configuration LP is to have variables for each time t describing the snapshot of the vehicles' routes up to time t. Different LPs arise depending on whether the snapshot is taken for each individual vehicle, or is a global snapshot of the k vehicles' routes.

Let $\mathcal{P}_t^i$ and $\mathcal{T}_t^i$ denote respectively the collection of all (simple) paths and trees rooted at r_i of length at most t. In our first configuration LP, we introduce a variable $z_{P,t}^i$ for every time t and path $P \in \mathcal{P}_t^i$ that

indicates if P is the path used to visit the nodes on vehicle i's route having latency at most t; that is, $z^i_{P,t}$ denotes if P is the portion of vehicle i's route up to time t. We also have variables $x^i_{v,t}$ to denote if node v is visited at time t by the route originating at root r_i.

$$\begin{aligned}
\min \quad & \sum_{v,t,i} t x^i_{v,t} && && (\mathrm{LP}_{\mathcal{P}}) \\
\text{s.t.} \quad & \sum_{t,i} x^i_{v,t} \geq 1 && \forall v && (1) \\
& \sum_{P \in \mathcal{P}^i_t} z_{P,t} \leq 1 && \forall t, i && (2) \\
& \sum_{P \in \mathcal{P}^i_t : v \in P} z^i_{P,t} \geq \sum_{t' \leq t} x^i_{v,t'} && \forall v, t, i && (3) \\
& x, z \geq 0.
\end{aligned}$$

$$\begin{aligned}
\max \quad & \sum_{v} \alpha_v - \sum_{t,i} \beta^i_t && && (\mathrm{D}) \\
\text{s.t.} \quad & \alpha_v \leq t + \sum_{t' \geq t} \theta^i_{v,t'} && \forall v, t, i && (4) \\
& \sum_{v \in P} \theta^i_{v,t} \leq \beta^i_t && \forall t, i, P \in \mathcal{P}_t && (5) \\
& \alpha, \beta, \theta \geq 0. && && (6)
\end{aligned}$$

Constraint (1) encodes that every non-root node must be visited by some vehicle at some time; (2) and (3) encode that at most one path corresponds to the portion of vehicle i's route up to time t, and that this path must visit every node v visited at any time $t' \leq t$ by vehicle i. Note that these enforce that $x^i_{v,t} = 0$ if $t < c_{r_i v}$. We remark that in the single-depot case, the splitting of the k paths into one path per vehicle is immaterial, and so ($\mathrm{LP}_{\mathcal{P}}$) becomes equivalent to the configuration LP in [10] for k-MLP that involves a single set of $x_{v,t}$ and $z_{P,t}$ variables for each time t.

In order to solve ($\mathrm{LP}_{\mathcal{P}}$), we consider the dual LP (D), which has exponentially many constraints. Separating over constraints (5) involves solving a (rooted) path-orienteering problem: for every t, given rewards $\{\theta^i_{u,t}\}_{u \in V}$, where we set $\theta^i_{u,t} = 0$ for $u \in R$, we want to determine if there is a path P rooted at r_i of length at most t that gathers reward more than β^i_t. In unweighted orienteering, all node rewards are 0 or 1. A (ρ, γ)-{path, tree} approximation algorithm for the path-orienteering problem is an algorithm that always returns a {path, tree} rooted at r_i of length at most γ(length bound) that gathers reward at least (optimum reward)$/\rho$.

As shown in [10], one can use such approximation algorithms to obtain an approximate solution to ($\mathrm{LP}_{\mathcal{P}}$) (and similar configuration LPs), where the notion of approximation involves bounded violation of the constraints and moving to a "tree version" of ($\mathrm{LP}_{\mathcal{P}}$). In the tree version of a configuration LP such as ($\mathrm{LP}_{\mathcal{P}}$), the only change is that configurations are defined in terms of trees instead of paths. Specifically, define the tree-version of ($\mathrm{LP}_{\mathcal{P}}$), denoted ($\mathrm{LP}_{\mathcal{T}}$), to be the analogue where we have variables $z^i_{Q,t}$ for every $Q \in \mathcal{T}^i_t$, and we replace all occurrences of $z^i_{P,t}$ in ($\mathrm{LP}_{\mathcal{P}}$) with $z^i_{Q,t}$.

Let $\left(\mathrm{LP}^{(a)}_{\mathcal{P}}\right)$ be ($\mathrm{LP}_{\mathcal{P}}$) where we replace each occurrence of $\mathcal{P}^i_t$ in constraints (2), (3) by $\mathcal{P}^i_{at}$, and the RHS of (2) is now a. Let $\left(\mathrm{LP}^{(a)}_{\mathcal{T}}\right)$ be defined analogously. Let $OPT_{\mathcal{P}}$ be the optimal value of ($\mathrm{LP}_{\mathcal{P}}$) (i.e., $\left(\mathrm{LP}^{(1)}_{\mathcal{P}}\right)$). Chaudhuri et al. [11] give a $(1, 1+\epsilon)$-tree approximation for unweighted orienteering, which yields (via suitably scaling and rounding the node rewards) a $(1+\epsilon, 1+\epsilon)$-tree approximation for weighted orienteering. Utilizing this and mimicking the arguments in [10] yields Lemma 4.1, which combined with our rounding procedure in Section 5 yields an $(8.497+\varepsilon)$-approximation algorithm for multi-depot k-MLP (Theorem 5.1).

Lemma 4.1. *For any $\epsilon > 0$, we can compute a feasible solution to $\left(LP^{(1+\epsilon)}_{\mathcal{T}}\right)$ of cost at most $OPT_{\mathcal{P}}$ in time* $\mathrm{poly}\left(\textit{input size}, \frac{1}{\epsilon}\right)$.

A stronger configuration LP. We now describe a stronger LP ($\mathrm{LP2}_{\mathcal{P}}$) that sheds further light on the power of LP-relaxations for minimum-latency problems. We prove that the integrality gap of ($\mathrm{LP2}_{\mathcal{P}}$) is at most $\mu^* < 3.5912$ by giving an efficient rounding procedure (Theorem 4.2 (ii)). We do not know how to leverage

this to obtain an efficient μ^*-approximation for multi-depot k-MLP, since we do not know how to solve (LP2$_\mathcal{P}$) efficiently, even in the approximate sense of Lemma 4.1. But for $k = 1$, (LP2$_\mathcal{P}$) coincides with (LP$_\mathcal{P}$) (and the configuration LP in [10]), so we can use Lemma 4.1 to approximately solve (LP2$_\mathcal{P}$). We also show that $OPT_{\text{LP2}_\mathcal{P}}$ is at least the value of the (k, ℓ)-bottleneck-stroll lower bound, BNSLB. Combined with Theorem 2.3, this provides another proof that the integrality gap of (LP2$_\mathcal{P}$) is at most μ^*; this also shows that the integrality gap of (LP2$_\mathcal{P}$) is at most 3.03 when $k = 1$.

In the new LP, a configuration for time t is the global snapshot of the k vehicles' routes up to time t; that is, it is the k-tuple formed by the portions of the k vehicles' routes up to time t. Formally, a configuration $\vec{P}$ for time t is a tuple $(P_1, \ldots, P_k)$ where each P_i is rooted at r_i and has length at most t. We say that v is covered by $\vec{P}$, and denote this by $v \in \vec{P}$, to mean that $v \in \bigcup_i V(P_i)$. Let $\mathcal{P}_t$ denote the collection of all configurations for time t. This yields the following LP whose constraints encode that every non-root node must be covered, there is at most one configuration for each time t, and this configuration must cover every node v whose latency is at most t.

$$\begin{aligned}
\min \quad & \sum_{v,t} t x_{v,t} && && \text{(LP2}_\mathcal{P}\text{)}\\
\text{s.t.} \quad & \sum_{t} x_{v,t} \geq 1 && \forall v && (7)\\
& \sum_{\vec{P} \in \mathcal{P}_t} z_{\vec{P},t} \leq 1 && \forall t && (8)\\
& \sum_{\vec{P} \in \mathcal{P}_t : v \in \vec{P}} z_{\vec{P},t} \geq \sum_{t' \leq t} x_{v,t'} && \forall v, t && (9)\\
& x, z \geq 0.
\end{aligned}$$

As before, we may define a tree version of (LP2$_\mathcal{P}$) similarly to the way in which (LP$_\mathcal{T}$) is obtained from (LP$_\mathcal{P}$). Define a *tree configuration* $\vec{Q}$ for time t to be a tuple $(Q_1, \ldots, Q_k)$, where each Q_i is an r_i-rooted tree of cost at most t. As before, we say that v is covered by $\vec{Q}$ if $v \in \bigcup_i V(Q_i)$ denote this by $v \in \vec{Q}$. Let $\mathcal{T}_t$ denote the collection of all tree configurations for time t. In the tree-version of (LP2$_\mathcal{P}$), denoted (LP2$_\mathcal{T}$), we have variables $z_{\vec{Q},t}$ for every $\vec{Q} \in \mathcal{T}_t$, and we replace constraints (8), (9) by

$$\sum_{\vec{Q} \in \mathcal{T}_t} z_{\vec{Q},t} \leq 1 \qquad \forall t \qquad (10) \qquad\qquad \sum_{\vec{Q} \in \mathcal{T}_t : v \in \vec{Q}} z_{\vec{Q},t} \geq \sum_{t' \leq t} x_{v,t'} \qquad \forall v, t \qquad (11)$$

We define $\left(\text{LP2}_\mathcal{T}^{(a)}\right)$ to be (LP2$_\mathcal{T}$), where we replace $\mathcal{T}_t$ in (10), (11) with $\mathcal{T}_{at}$, and we replace the RHS of (10) with a. The following theorem suggests that (LP2$_\mathcal{P}$) may be quite powerful; we defer its proof to Section 8.

Theorem 4.2. *We have the following.*

(i) $OPT_{LP2_\mathcal{P}} \geq$ BNSLB *for every instance of multi-depot* k-MLP.

(ii) A solution to (LP2$_\mathcal{P}$)*, can be efficiently rounded to a feasible integer solution while increasing the cost by a factor of at most* $\mu^* < 3.5912$*. Furthermore, for any* $\epsilon \geq 0$*, a solution to* $\left(LP2_\mathcal{T}^{(1+\epsilon)}\right)$ *can be efficiently rounded to a feasible solution to multi-depot* k-MLP *while losing a factor of at most* $\frac{\mu^*}{1-\mu^*\epsilon}$.

(iii) When $k = 1$*, for any* $\epsilon > 0$*, we can compute a feasible solution to* $\left(LP2_\mathcal{T}^{(1+\epsilon)}\right)$ *of cost at most* $OPT_{LP2_\mathcal{P}}$ *in time* $\text{poly}\left(\textit{input size}, \frac{1}{\epsilon}\right)$.

4.2 A bidirected LP relaxation

The bidirected LP formulation is motivated by Theorem 3.1. As in Section 3, we bidirect the edges to obtain a digraph $D = (V, A)$ and set $c_a = c_{uv}$ for both $a = (u, v)$ and $a = (v, u)$. We use a to index the arcs in A.

Recall that v indexes nodes in $V \setminus R$, i indexes the k vehicles, and t indexes time units in $[\mathsf{T}]$. As before, we use variables $x^i_{v,t}$ to denote if node v is visited at time t by the route originating at root r_i. Directing the vehicles' routes away from their roots in a solution, $z^i_{a,t}$ indicates if arc a lies on the portion of vehicle i's route up to time t. We obtain the following LP.

$$\min \quad \sum_{v,t,i} t x^i_{v,t} \tag{LP3}$$

$$\text{s.t.} \quad \sum_{t,i} x^i_{v,t} \geq 1 \qquad \forall v; \qquad x^i_{v,t} = 0 \text{ if } c_{r_i v} < t \qquad \forall v,t,i \tag{12}$$

$$\sum_{a \in \delta^{\text{in}}(S)} z^i_{a,t} \geq \sum_{t' \leq t} x^i_{v,t'} \qquad \forall S \subseteq V \setminus \{r_i\}, v \in S, \quad \forall t \tag{13}$$

$$\sum_{a} c_a z^i_{a,t} \leq t \qquad \forall t, i \tag{14}$$

$$\sum_{a \in \delta^{\text{in}}(v)} z^i_{a,t} \geq \sum_{a \in \delta^{out}(v)} z^i_{a,t} \qquad \forall v, i \tag{15}$$

$$x, z \geq 0. \tag{16}$$

Constraints (12) ensure that every non-root node is visited at some time. Constraints (13)–(15) play the role of constraints (2), (3) in (LP$_{\mathcal{P}}$): (13) ensures that the portion of a vehicle's route up to time t must visit every node visited by that vehicle by time t, (14) ensures that this route indeed has length at most t, and finally (15) seeks to encode that the route forms a path. (Note that constraints (15) are clearly valid, and one could also include the constraints $\sum_{a \in \delta^{\text{out}}(r_i)} z^i_{a,t} \leq 1$ for all i, t.)

Assuming T is polynomially bounded, it is easy to design a separation oracle for the exponentially-many cut constraints (13); hence, one can solve (LP3) efficiently. We use this LP to obtain LP-relative guarantees for k-MLP (Section 6) which turn out to be quite versatile and extend easily to yield the same guarantees for various generalizations.

Intuitively, the difference between (LP$_{\mathcal{P}}$) and (LP3) boils down to the following. Consider the tree version of (LP$_{\mathcal{P}}$), (LP$_{\mathcal{T}}$). A solution to this LP specifies for each time t and vehicle i, a distribution over r_i-rooted trees that together cover nodes to the extent dictated by the x variables. Using Theorem 3.1, given a feasible solution (x, z) to (LP3), one can view z as also specifying for each time t and vehicle i a distribution over r_i-rooted trees covering nodes to the extents specified by the x variables. The difference however is that in the former case, *each tree in the support has length at most t*, whereas in the distribution obtained from (LP3) one only knows that the *expected length* of a tree is at most t.

Theorem 4.3 (Relationship between (LP$_{\mathcal{P}}$)–(LP3))**.** *We have the following. Recall that $OPT_{\mathcal{P}}$ is the optimal value of* (LP$_{\mathcal{P}}$).

(i) $OPT_{LP3} \leq OPT_{\mathcal{P}} \leq OPT_{LP2_{\mathcal{P}}}$ *for every instance of multi-depot k-*MLP*. When $k = 1$, we have* $OPT_{\mathcal{P}} = OPT_{LP2_{\mathcal{P}}}$.

*(ii) For multi-depot k-*MLP*, we can efficiently compute a solution of cost at most* $(8.4965 + \varepsilon) OPT_{\mathcal{P}}$ *for any* $\varepsilon > 0$ *(also Theorem 5.1).*

*(iii) For single-depot k-*MLP*, we can efficiently compute a solution of cost at most* $(2\mu^* + \varepsilon) OPT_{LP3}$ *for any* $\varepsilon > 0$*, where* $\mu^* < 3.5912$ *(also Theorem 6.1).*

(iv) When $k = 1$, we can efficiently compute a solution of cost at most $(\mu^* + \varepsilon) OPT_{LP3}$ *for any* $\varepsilon > 0$ *(also Corollary 6.4).*

Proof. Parts (ii)–(iv) are simply restatements of the indicated theorems, whose proofs appear in the corresponding sections. We focus on proving part (i).

For a k-tuple $\vec{P} = (P_1, \ldots, P_k)$, we use $\vec{P}(i)$ to denote P_i. Let (x, z) be a feasible solution to (LP2$_{\mathcal{P}}$). We may assume that constraints (7), (9) hold with equality for all v and t since we can always shortcut paths past nodes without increasing their length. We may also assume that if $z_{\vec{P},t} > 0$ for $\vec{P} = (P_1, \ldots, P_k)$, then the any two P_is are node-disjoint unless they originate from the same root node, in which case this root is the only node they share. We map (x, z) to a feasible solution (x', z') to (LP$_{\mathcal{P}}$) by setting $z'^i_{P_i,t} = \sum_{\vec{P} \in \mathcal{P}_t : \vec{P}(i) = P_i} z_{\vec{P},t}$ for all i, t and $x'^i_{v,t} = \sum_{P \in \mathcal{P}^i_t : v \in P} z'^i_{P,t} - \sum_{P \in \mathcal{P}^i_{t-1} : v \in P} z'^i_{P,t-1}$ for all i, v, t, where we define $z'^i_{P,0} = 0$ for all P for notational convenience. It is easy to verify that (x', z') is feasible for (LP$_{\mathcal{P}}$). It's objective value is

$$\begin{aligned}
\sum_{v,t,i} t x'^i_{v,t} &= \sum_{v,i} \sum_{t=1}^{T} t \Big(\sum_{P \in \mathcal{P}^i_t : v \in P} z'^i_{P,t} - \sum_{P \in \mathcal{P}^i_{t-1} : v \in P} z'^i_{P,t-1} \Big) \\
&= \sum_{v,i} \Big(\mathsf{T} \sum_{P \in \mathcal{P}^i_{\mathsf{T}} : v \in P} z'^i_{P,\mathsf{T}} - \sum_{t=1}^{\mathsf{T}-1} \sum_{P \in \mathcal{P}^i_t : v \in P} z'^i_{P,t} \Big) = \sum_{v} \Big(\mathsf{T} - \sum_{t=1}^{\mathsf{T}-1} \sum_{\vec{P} \in \mathcal{P}_t : v \in \vec{P}} z_{\vec{P},t} \Big) \qquad (17) \\
&= \sum_{v} \Big(\mathsf{T} - \sum_{t=1}^{\mathsf{T}-1} \sum_{t' \le t} x_{v,t'} \Big) = \sum_{v} \Big(\mathsf{T} - \sum_{t'=1}^{\mathsf{T}} (\mathsf{T} - t') x_{v,t'} \Big) = \sum_{v,t'} t' x_{v,t'} \qquad (18)
\end{aligned}$$

The second equality in (17) holds because $\sum_i \sum_{P \in \mathcal{P}^i_t : v \in P} z'^i_{P,t} = \sum_i \sum_{\vec{P} \in \mathcal{P}_t : v \in \vec{P}(i)} z_{\vec{P},t} = \sum_{\vec{P} \in \mathcal{P}_t : v \in \vec{P}} z_{\vec{P},t}$ since the paths comprising $\vec{P}$ do not share any non-root nodes; when $t = T$, this term is 1 since (9) and (7) hold at equality. The first and third equalities in (18) follow again from the fact that (9) and (7) hold at equality. It follows that $OPT_{\mathcal{P}} \le OPT_{\text{LP2}_{\mathcal{P}}}$.

Let (x, z) be a feasible solution to (LP$_{\mathcal{P}}$). It is easy to see that if we direct each path in the support of z away from its root and set $z'^i_{a,t} = \sum_{P \in \mathcal{P}^i_t : a \in P} z^i_{P,t}$, then (x, z') is feasible for (LP3). Hence, $OPT_{\text{LP3}} \le OPT_{\mathcal{P}}$. ■

5 An LP-rounding 8.497-approximation algorithm for multi-depot k-MLP

We now prove the following theorem. Our approximation ratio of 8.497 improves upon the previous-best 12-approximation [12, 11] and matches the previous-best approximation for single-depot k-MLP [14].

Theorem 5.1. *For any $\varepsilon > 0$, we can compute a multi-depot-k-MLP solution of cost at most $(8.4965 + \varepsilon) \cdot OPT_{\mathcal{P}}$ in time $\text{poly}\big(\text{input size}, \frac{1}{\varepsilon}\big)$. Thus, the integrality gap of (LP$_{\mathcal{P}}$) is at most 8.4965.*

Our algorithm is quite simple to describe. Let (x, z) be the feasible solution to $\big(\text{LP}^{(1+\epsilon)}_{\mathcal{T}}\big)$ returned by Lemma 4.1, where we fix ϵ later. We then choose time points that form a geometric sequence and do the following for each time point t. For every $i = 1, \ldots, k$, we sample a random tree from the distribution $\{z^i_{Q,t}\}_{Q \in \mathcal{T}^i_{(1+\epsilon)t}}$, double and shortcut it to form a cycle and traverse this cycle in a random direction to obtain a tour. For every i, we concatenate the tours obtained for i for each of the time points. We now describe the rounding procedure in detail and proceed to analyze it.

Algorithm 1. Given: a fractional solution (x, z) of cost at most $OPT_{\mathcal{P}}$ returned by Lemma 4.1.

M1. Let $\kappa = 1 + \epsilon$, and $1 < c < e$ be a constant that we will fix later. Let $h = c^{\Gamma}$ be a random offset, where Γ is chosen uniformly at random from $[0, 1)$. For notational convenience, define $z^i_{Q,t}$ for all $t \geq 1$, i, $Q \in \mathcal{T}^i_{\kappa t}$ as follows: set $z^i_{Q,t} = z^i_{Q,\lfloor t \rfloor}$ if $\lfloor t \rfloor \leq \mathsf{T}$ and $z^i_{Q,t} = z^i_{Q,\mathsf{T}}$ otherwise. Define $t_j = hc^j$ for all $j \geq 0$.

M2. Repeatedly do the following for $j = 0, 1, 2, \ldots$ until every non-root node is covered (by some tour). For every $i = 1, \ldots, k$, choose independently a random tree Q from the distribution $\{z^i_{Q,t_j}/\kappa\}_{Q \in \mathcal{T}^i_{\kappa t_j}}$. Double and shortcut Q to get a cycle, and traverse this cycle clockwise or counterclockwise with probability $\frac{1}{2}$ to obtain a tour $Z_{i,j}$.

M3. For every $i = 1, \ldots, k$, concatenate the tours $Z_{i,0}, Z_{i,1}, \ldots$ to obtain the route for vehicle i.

Analysis. The analysis hinges on showing that for every iteration j of step M2, the probability $p_{v,j}$ that a node v is not covered by the end of iteration j can be bounded (roughly speaking) in terms of the total extent to which v is not covered by (x, z) by time t_j (Lemma 5.3). Substituting this into the expression bounding the expected latency of v in terms of the $p_{v,j}$s (part (iii) of Claim 5.2), we obtain that by suitably choosing the constant c, the expected latency of v is roughly $8.497 \cdot \mathsf{lat}_v$, where $\mathsf{lat}_v := \sum_{t,i} t x^i_{v,t}$ (Lemma 5.4). This proves that the algorithm returns a solution of cost roughly $8.497 \cdot OPT_{\mathcal{P}}$. However, it is not clear if the algorithm as stated above has polynomial running time. But since Lemma 5.3 implies that $p_{v,j}$ decreases geometrically with j, one can terminate step M2 after a polynomial number of iterations and cover the remaining uncovered nodes incurring latency at most T for each such node. This increases the expected cost by at most $\varepsilon OPT_{\mathcal{P}}$ but ensures polynomial running time; see Remark 5.5.

Let $t_{-1} = 0$, and define $\Delta_j := t_j - t_{j-1}$ for all $j \geq 0$. For $q \geq 1$, define $\sigma(q)$ to be the smallest t_j that is at least q. Consider a non-root node v. We may assume that $\sum_{i,t} x^i_{v,t} = 1$. Define $p_{v,j} = 1$ for all $j < 0$. Define $y^i_{v,t} := \sum_{t' \leq t} x^i_{v,t'}$ and $o'_{v,j} := 1 - \sum_i y^i_{v,t_j}$; define $o'_{v,j} = 1$ for all $j < 0$. Define $\mathsf{lat}'_v := \sum_{j \geq 0} o'_{v,j-1} \Delta_j$. Let L_v denote the random latency of node v in the solution constructed. Note that the t_js, and hence, $\sigma(q)$, the $o'_{v,j}$s and lat'_v are random variables depending only on the random offset h. For a fixed offset h, we use $\mathrm{E}^h[.]$ to denote the expectation with respect to all other random choices, while $\mathrm{E}[.]$ denotes the expectation with respect to all random choices.

Claim 5.2. *For any node v, we have: (i) $\mathsf{lat}'_v = \sum_{t,i} \sigma(t) x^i_{v,t}$; (ii) $\mathrm{E}[\mathsf{lat}'_v] = \frac{c-1}{\ln c} \cdot \mathsf{lat}_v$; and (iii) $\mathrm{E}^h[L_v] \leq \frac{\kappa(c+1)}{c-1} \cdot \sum_{j \geq 0} p_{v,j-1} \Delta_j$ for any fixed h.*

Proof. Part (i) follows from the same kind of algebraic manipulation as used in the proof of part (i) of Theorem 4.2. We have

$$\sum_{t,i} \sigma(t) x^i_{v,t} = \sum_{j \geq 0} t_j \Big(\sum_{t=t_{j-1}+1}^{t_j} \sum_i x^i_{v,t} \Big) = \sum_{j \geq 0} \Big(\sum_{d=0}^{j} \Delta_d \Big) \Big(\sum_{t=t_{j-1}+1}^{t_j} \sum_i x^i_{v,t} \Big)$$
$$= \sum_{d \geq 0} \Delta_d \Big(\sum_{j \geq d} \sum_{t=t_{j-1}+1}^{t_j} \sum_i x^i_{v,t} \Big) = \sum_{d \geq 0} \Delta_d o'_{v,d-1}.$$

Part (ii) follows from part (i) since we show that $\mathrm{E}[\sigma(q)] = \frac{c-1}{\ln c} \cdot q$ for all $q \geq 1$. Suppose $q \in [c^j, c^{j+1})$ for some integer $j \geq 0$. Then

$$\mathrm{E}[\sigma(q)] = \int_0^{\log_c q - j} c^{y+j+1} dy + \int_{\log_c q - j}^{1} c^{y+j} dy = \frac{1}{\ln c} \cdot \left(c^{\log_c q + 1} - c^{j+1} + c^{j+1} - c^{\log_c q} \right) = \frac{c-1}{\ln c} \cdot q.$$

For part (iii), say that node v is covered in iteration $j \geq 0$ if j is the smallest index such that $v \in \bigcup_i V(Z_{i,j})$. By definition, the probability of this event is $p_{v,j-1} - p_{v,j}$, and in this case the latency of v is at most $\kappa(2t_0 + 2t_1 + \ldots + 2t_{j-1} + t_j) \leq \frac{\kappa(c+1)}{c-1} \cdot t_j$. So $\mathrm{E}^h[L_v] \leq \frac{\kappa(c+1)}{c-1} \sum_{j \geq 0} (p_{v,j-1} - p_{v,j}) t_j = \frac{\kappa(c+1)}{c-1} \sum_{j \geq 0} p_{v,j-1}(t_j - t_{j-1})$. ∎

Lemma 5.3. $p_{v,j} \le \big(1 - e^{-1/\kappa}\big) o'_{v,j} + e^{-1/\kappa} p_{v,j-1}$ *for all* $j \ge -1$, *and all* v.

Proof. For $j = -1$, the inequality holds since $o'_{v,-1} = 1 = p_{v,-2}$. Suppose $j \ge 0$. We have $p_{v,j} \le p_{v,j-1} \prod_i \big(1 - \frac{y^i_{v,t_j}}{\kappa}\big)$ since the probability that v is visited by the i-th tour in iteration j is $\sum_{Q \in \mathcal{T}^i_{\kappa t_j} : v \in Q} z^i_{Q,t_j}/\kappa \ge y^i_{v,t_j}/\kappa$. We have that $\prod_{i=1}^k \big(1 - \frac{y^i_{v,t_j}}{\kappa}\big)$ is at most

$$\left(1 - \frac{\sum_i y^i_{v,t_j}}{\kappa k}\right)^k = \left(1 - \frac{1 - o'_{v,j}}{\kappa k}\right)^k \le \left(1 - \tfrac{1}{\kappa k}\right)^k + \left(1 - \left(1 - \tfrac{1}{\kappa k}\right)^k\right) o'_{v,j} \le e^{-1/\kappa} + \big(1 - e^{-1/\kappa}\big) o'_{v,j}.$$

The first inequality follows since the geometric mean is at most the arithmetic mean; the second follows since $f(b) = \big(1 - \frac{1-b}{\kappa k}\big)^k$ is a convex function of b; the final inequality follows since $o'_{v,j} \le 1$ and $\big(1 - \frac{1}{\kappa k}\big)^k \le e^{-1/\kappa}$. Plugging the above bound into the inequality for $p_{v,j}$ yields the lemma. ■

Lemma 5.4. $\mathrm{E}[L_v] \le \frac{\kappa(c+1)(1-e^{-1/\kappa})}{(\ln c)(1-ce^{-1/\kappa})} \cdot \mathsf{lat}_v$ *for all* v.

Proof. Fix an offset h. We have $\frac{c-1}{\kappa(c+1)} \cdot \mathrm{E}^h[L_v] \le A := \sum_{j \ge 0} p_{v,j-1} \Delta_j$ by Claim 5.2 (iii). Using Lemma 5.3, we obtain that $A \le \sum_{j \ge 0} \big(1 - e^{-1/\kappa}\big) o'_{v,j-1} \Delta_j + e^{-1/\kappa} \sum_{j \ge 0} p_{v,j-2} \Delta_j = \big(1 - e^{-1/\kappa}\big) \mathsf{lat}'_v + c e^{-1/\kappa} A$, where the equality follows since $\Delta_0 + \Delta_1 = c\Delta_0$, and $\Delta_j = c\Delta_{j-1}$ for all $j \ge 2$, and so $\sum_{j \ge 0} p_{v,j-2} \Delta_j = cA$. So $A \le \frac{1 - e^{-1/\kappa}}{1 - ce^{-1/\kappa}} \mathsf{lat}'_v$. Taking expectation with respect to the random offset h, and plugging in the bound for $\mathrm{E}[\mathsf{lat}'_v]$ in part (ii) of Claim 5.2 yields the lemma. ■

Taking $c = 1.616$, for any $\varepsilon > 0$, we can take $\epsilon > 0$ suitably small so that $\frac{\kappa(c+1)(1-e^{-1/\kappa})}{(\ln c)(1-ce^{-1/\kappa})} \le \frac{(c+1)(1-e^{-1})}{(\ln c)(1-ce^{-1})} + \varepsilon \le 8.4965 + \varepsilon$. This completes the proof of Theorem 5.1.

Remark 5.5. If we truncate step M2 to $N = D + \kappa \ln\big(\frac{n\mathsf{T}}{\varepsilon}\big)$ iterations, where $t_D = \sigma(\mathsf{T})$, then by Lemma 5.3, $p_{v,N} \le e^{-(N-D)/\kappa} \le \frac{\varepsilon}{n\mathsf{T}}$ since $o'_{v,J} = 0$. Each remaining uncovered node can be covered incurring latency at most T (since T is a certifiable upper bound). This adds at most $\varepsilon \le \varepsilon OPT_{\mathcal{P}}$ to the expected cost of the solution, but ensures polynomial running time.

6 A 7.183-approximation algorithm for (single-depot) k-MLP

We now describe algorithms for k-MLP having approximation ratios essentially $2\mu^* < 7.183$. This guarantee can be obtained both by rounding the bidirected LP (LP3) and via more combinatorial methods. The LP-rounding algorithm is slightly easier to describe, and the analysis extends easily to the generalizations considered in Section 7. But it is likely less efficient than the combinatorial algorithm, and its guarantee is slightly weaker, $2\mu^* + \varepsilon$, due to the fact that we need to solve the time-indexed formulation (LP3) either by ensuring that T is polynomially bounded, or via the alternative method sketched in Section 7, both of which result in a $(1+\varepsilon)$-factor degradation in the approximation. We describe the LP-rounding algorithm first (Section 6.1) and then the combinatorial algorithm (Section 6.2).

6.1 The LP-rounding algorithm

We prove the following theorem. Recall that $D = (V, A)$ is the digraph obtained by bidirecting G.

Theorem 6.1. *Any solution* (x, z) *to* (LP3) *can be rounded to a* k-MLP*-solution losing a factor of at most* $2\mu^* < 7.1824$. *Thus, for any* $\varepsilon > 0$, *we can compute a* k-MLP*-solution of cost at most* $(2\mu^* + \varepsilon) OPT_{LP3}$ *in time* $\mathrm{poly}\big(\textit{input size}, \ln(\frac{1}{\varepsilon})\big)$.

The rounding algorithm follows the familiar template of finding a collection of tours with different node coverages and stitching them together using a concatenation graph. Let (x, z) be a feasible solution to (LP3). Let $x'_{v,t} = \sum_i x^i_{v,t}$ and $z'_{a,t} = \sum_i z^i_{a,t}$. We will in fact only work with (x', z'). (This also implies that we obtain the same $2\mu^*$-guarantee with respect to an even weaker bidirected LP where we aggregate the k vehicles' routes and use a single set of $x_{v,t}$ and $z_{a,t}$ variables for all v, a, t.) For notational convenience, define $x'_{r,0} = 1$, $x'_{r,t} = 0$ for all $t > 0$, and $x'_{v,0} = 0$ for all $v \neq r$. To give some intuition behind the proof of Theorem 6.1, the following lemma will be useful. The proof involves simple algebraic manipulation, and is deferred to the end of this section.

Lemma 6.2. *Suppose for every time $t = 0, 1, \ldots, \mathsf{T}$, we have a random variable $(N_t, Y_t) \in [1, n] \times \mathbb{R}$ such that $(N_0, Y_0) = (1, 0)$ with probability 1, and $\mathrm{E}[N_t] \geq \sum_{u \in V} \sum_{t'=0}^{t} x'_{u,t'}$, $\mathrm{E}[Y_t] \leq \alpha t$, for all $t \in [\mathsf{T}]$. Let f be the lower-envelope curve of $\bigcup_{t=0}^{\mathsf{T}}$(support of (N_t, Y_t)). Then, $\int_1^n f(x)dx \leq \alpha \sum_{u \in V, t \in [\mathsf{T}]} t x'_{u,t}$.*

Lemma 6.2 coupled with Corollary 2.2 imply that if one could efficiently compute for each time t, a random collection of k trees rooted at r such that (a) their union covers in expectation at least $\sum_{u \in V} \sum_{t'=0}^{t} x'_{u,t'} = 1 + \sum_{v \neq r, t' \in [t]} x'_{v,t'}$ nodes, and (b) the expected maximum length of a tree in the collection is at most t, then we would achieve a μ^*-approximation by mimicking the proof of part (i) of Theorem 2.3. We do not quite know how to achieve this. However, as in the proof of Theorem 3.2, applying Theorem 3.1 to (a scaled version of) $(z'_{a,t})_{a \in A}$, we can efficiently find *one* random r-rooted tree that in expectation has cost at most kt and covers at least $\sum_{u \in V} \sum_{t'=0}^{t} x'_{u,t'}$ nodes. This is the chief source of our improvement over the 8.497-approximation in [14]: we achieve the target coverage of the k vehicles (which in our case is determined by an LP) whereas [14] sequentially find separate tours for each vehicle, which succeeds in covering only a constant-fraction of the target number of nodes (which in their case is determined by the integer optimal solution).

The flip side is that we need to do slightly more work to convert the object computed into a (random) collection of k low-cost tours containing r. To convert a rooted tree, we Eulerify it, break the resulting cycle into k segments, and attach each segment to r. Thus, for each time t, we obtain a random collection of k trees rooted at r satisfying property (a) above, and a relaxed form of (b): the expected maximum length of a tree in the collection is at most $2t$. Thus, we obtain a solution of cost at most $2\mu^*$ times the cost of (x, z). We now describe the rounding algorithm in more detail and proceed to analyze it.

Algorithm 2. The input is a feasible solution (x, z) to (LP3). Let $x'_{v,t} = \sum_i x^i_{v,t}$, $z'_{a,t} = \sum_i z^i_{a,t'}$ for all v, a, t.

R1. Initialize $C \leftarrow \{(1, 0)\}$, $\mathcal{Q} \leftarrow \emptyset$. Let K be such $Kz'_{a,t}$ is an integer for all a, t. For $t \in [\mathsf{T}]$, define $S(t) = \{u \in V : \sum_{t'=0}^{t} x'_{u,t'} > 0\}$. (Note that $r \in S(t)$ for all $t > 0$.)

R2. For all $t = 1, \ldots, \mathsf{T}$, do the following. Apply Theorem 3.1 on the digraph D with edge weights $\{Kz'_{a,t}\}_{a \in A}$ and integer K (and root r) to obtain a weighted arborescence family $(\gamma_1, Q^t_1), \ldots, (\gamma_q, Q^t_q)$. For each arborescence Q^t_ℓ in the family, which we view as a tree, add the point $\left(|V(Q^t_\ell) \cap S(t)|, \frac{2c(Q^t_\ell)}{k} + 2t\right)$ to C, and add the tree Q^t_ℓ to $\mathcal{Q}$.

R3. For all $\ell = 1, \ldots, n$, compute $s_\ell = f(\ell)$, where $f : [1, n] \mapsto \mathbb{R}_+$ is the lower-envelope curve of C. We show in Lemma 6.3 that for every corner point $\big(\ell, f(\ell)\big)$ of f, there is some tree $Q^*_\ell \in \mathcal{Q}$ and some time t^*_ℓ such that $\ell = |V(Q^*_\ell) \cap S(t^*_\ell)|$, $f(\ell) = \frac{2c(Q^*_\ell)}{k} + 2t^*_\ell$, and $\max_{v \in Q^*_\ell \cap S(t^*_\ell)} c_{rv} \leq t^*_\ell$.

R4. Find a shortest $1 \rightsquigarrow n$ path P_C in the concatenation graph $CG(s_1, \ldots, s_n)$.

R5. For every node $\ell > 1$ on P_C, do the following. Double and shortcut Q^*_ℓ to obtain a cycle. Remove nodes on this cycle that are not in $S(t^*_\ell)$ by shortcutting past such nodes. Break this cycle into k segments, each of length at most $2c(Q^*_\ell)/k$ and add edges connecting the first and last vertex of each segment to r. This yields a collection of k cycles; traverse each resulting cycle in a random direction to obtain a collection of k tours $Z_{1,\ell}, \ldots, Z_{k,\ell}$.

R6. For every $i = 1, \ldots, k$, concatenate the tours $Z_{i,\ell}$ for nodes ℓ on P_C to obtain vehicle i's route.

Analysis. We first prove Lemma 6.2. Lemma 6.3 utilizes this to bound $\int_1^n f(x)dx$, and shows that corner points of f satisfy the properties stated in step R3. The latter allows us to argue that the solution returned has cost at most the length of P_C in the concatenation graph. Combining these facts with Corollary 2.2 yields the $2\mu^*$ approximation ratio and completes the proof of Theorem 6.1.

Proof of Lemma 6.2. Note that f is strictly increasing: for $x', x \in [1, n]$ with $x' < x$, since $x' = \frac{x'-1}{x-1} \cdot x + \frac{x-x'}{x-1} \cdot 1$ and $f(1) = 0$, we have $f(x') \leq \frac{x'-1}{x-1} f(x) < f(x)$. So we can write

$$\frac{\int_1^n f(x)dx}{\alpha} = \int_1^n \Big(\int_0^{f(x)/\alpha} dy \Big) dx = \int_0^{f(n)/\alpha} dy \Big(\int_{f^{-1}(\alpha y)}^n dx \Big) = \int_0^{f(n)/\alpha} \big(n - f^{-1}(\alpha y)\big) dy. \quad (19)$$

Note that $f(n) \leq \alpha \mathsf{T}$. For any $t = 0, 1, \ldots, \mathsf{T}$, the point $(\mathrm{E}[N_t], \mathrm{E}[Y_t]) = \mathrm{E}[(N_t, Y_t)]$ lies in the convex hull of the support of (N_t, Y_t), and so $f(\mathrm{E}[N_t]) \leq \mathrm{E}[Y_t] \leq \alpha t$ and hence, $\mathrm{E}[N_t] \leq f^{-1}(\alpha t)$. So we can bound (19) by

$$\begin{aligned} \sum_{t=1}^{\mathsf{T}} \int_{t-1}^{t} \big(n - f^{-1}(\alpha y)\big) dy &\leq \sum_{t=1}^{\mathsf{T}} \big(n - f^{-1}(\alpha(t-1))\big) \\ &\leq \sum_{t=1}^{\mathsf{T}} \Big(n - \sum_{u \in V} \sum_{t'=0}^{t-1} x'_{u,t'}\Big) \leq \sum_{t=1}^{\mathsf{T}} \sum_{u \in V, t' \geq t} x'_{u,t'} = \sum_{u \in V} \sum_{t'=1}^{\mathsf{T}} t' x'_{u,t'}. \end{aligned}$$ ■

Lemma 6.3. *(i) $\int_1^n f(x)dx \leq 4 \sum_{u \in V, t \in [\mathsf{T}]} t x'_{u,t}$. (ii) If $\big(\ell, f(\ell)\big)$ is a corner point of f, then there is a tree Q_ℓ^* and time t_ℓ^* satisfying the properties stated in step S3.*

Proof. Consider any $t \in [\mathsf{T}]$. The weighted arborescence family $(\gamma_1, Q_1^t), \ldots, (\gamma_q, Q_q^t)$ yields a distribution over arborescences, where we pick arborescence Q_ℓ^t with probability γ_ℓ / K. Let N_t and Y_t be the random variables denoting $|V(Q_\ell^t) \cap S(t)|$ and $\frac{2c(Q_\ell^t)}{k} + 2t$ respectively. Define $\lambda_D(r, r) = K$. By Theorem 3.1, we have $\mathrm{E}[c(Q_\ell^t)] \leq \sum_a c_a z'_{a,t} \leq kt$, so $\mathrm{E}[Y_t] \leq 4t$; also, $\mathrm{E}[N_t] \geq \sum_{u \in S(t)} \lambda_D(r, u)/K \geq \sum_{u \in S(t)} \sum_{t'=0}^{t} x'_{u,t'} = \sum_{u \in V} \sum_{t'=0}^{t} x'_{u,t'}$. So the random variables $(N_t, Y_t)_{t \in [\mathsf{T}]}$ and $(N_0, Y_0) = (1, 0)$ and the curve f satisfy the conditions of Lemma 6.2 with $\alpha = 4$, and its conclusion proves part (i).

For part (ii), corner points of f are points of C. So if $\big(\ell, f(\ell)\big) \in C$ is a corner point that was added to C in iteration t of step R2, then set $t_\ell^* = t$, and Q_ℓ^* to be the tree added in this iteration. By definition, we have $|V(Q_\ell^*) \cap S(t_\ell^*)| = \ell$, $f(\ell) = \frac{2c(Q_\ell^*)}{k} + 2t_\ell^*$. Also $\max_{v \in Q_\ell^* \cap S(t_\ell^*)} c_{rv} \leq t_\ell^*$, since v can only lie in $S(t_\ell^*)$ if $c_{rv} \leq t_\ell^*$, due to constraint (12). ■

Proof of Theorem 6.1. We claim that the solution returned by Algorithm 2 has cost at most the length of P_C in the concatenation graph $CG(s_1, \ldots, s_n)$. Combining this with Corollary 2.1 and part (i) of Lemma 6.5, we obtain that the total latency is at most $\frac{\mu^*}{2} \int_1^n f(x)dx \leq 2\mu^* \sum_{v,t} t x'_{v,t} = 2\mu^* \sum_{v,i,t} t x^i_{v,t}$.

The proof of the claim is similar to the one in [18] for single-vehicle MLP. Consider an edge (o, ℓ) of P_C. By Theorem 2.1, $\big(\ell, f(\ell)\big)$ is a corner point of f, so there exist Q_ℓ^*, t_ℓ^* satisfying the properties stated in step R3. It follows that when we transform Q_ℓ^* into k cycles containing only nodes of $S(t_\ell^*)$, each cycle has length at most $f(\ell) = s_\ell$.

Suppose inductively that we have covered at least o nodes by the partial solution constructed by stitching tours corresponding to the nodes on P_C up to and including o. Consider the additional contribution to the total latency when we concatenate tour $Z_{i,\ell}$ to vehicle i's current route, for $i = 1, \ldots, k$. The resulting partial solution covers at least ℓ nodes. A node covered in this step incurs additional latency at most $\frac{s_\ell}{2}$ since we traverse the cycle containing it (which has cost at most s_ℓ) in a random direction. A node that is still uncovered after this step incurs additional latency at most s_ℓ. There are at most $\ell - o$ new nodes

covered in this step, and at most $n-\ell$ uncovered nodes after this step, so the increase in total latency is at most $\frac{s_\ell}{2}(\ell-o)+s_\ell(n-\ell)=s_\ell\big(n-\frac{o+\ell}{2}\big)$, which is exactly the length of (o,ℓ) edge in $CG(s_1,\ldots,s_n)$. Therefore, by induction the total latency is at most the length of P_C in $CG(s_1,\ldots,s_n)$. ■

Corollary 6.4. *For single-vehicle* MLP*, any solution (x,z) to (LP3) can be rounded losing a factor of at most $\mu^*<3.5912$. Hence, for any $\varepsilon>0$, we can compute an* MLP*-solution of cost at most $(\mu^*+\varepsilon)\,OPT_{LP3}$ in time* $\mathrm{poly}\big($*input size*$,\frac{1}{\varepsilon}\big)$.

Proof. This follows from essentially Algorithm 2 and its analysis. The improvement comes because we no longer need to break up a tree into k tours. So in step R2, for each tree Q^t_ℓ in the weighted arborescence family obtained for time t, we add the point $\big(|V(Q^t_\ell)\cap S(t)|, 2c(Q^t_\ell)\big)$ to C, and in part(i) of Lemma 6.3, we have the stronger bound $\int_1^n f(x)dx\le 2\sum_{u\in V,t\in[\mathsf{T}]} tx'_{u,t}$, which yields the μ^* approximation. ■

6.2 The combinatorial approximation algorithm

The combinatorial algorithm for k-MLP follows a similar approach as the LP-rounding algorithm. The difference is that instead of using an LP to determine the target coverage of the k vehicles and maximum length of each vehicle's route, we now seek to match the target coverage and length bound of an optimal (k,ℓ)-bottleneck-stroll. Corollary 3.3 shows that one can efficiently find a rooted tree or bipoint tree that is at least as good (in terms of both total cost and node-coverage) as the optimal (k,ℓ)-bottleneck-stroll solution for all ℓ, and again this is where we score over the algorithm in [14]. Again, we need to convert the object computed into a collection of k tours, and Theorem 2.1 implies that a bipoint tree can be handled by handling the trees comprising it. We convert a tree into k tours as before, but to bound the cost of each resulting tour, we now need to "guess" the node furthest from r covered by an optimal (k,ℓ)-bottleneck stroll solution, and apply Corollary 3.3 with more-distant nodes removed; this ensures that each resulting tour has cost at most $4\cdot\mathsf{BNS}(k,\ell)$. Hence, mimicking the proof of Theorem 2.3 (i) shows that we obtain a solution of cost at most $2\mu^*\cdot\mathsf{BNSLB}<7.183\cdot\mathsf{BNSLB}$. The algorithm and analysis are very similar to that in Section 6.1.

Algorithm 3.

S1. Initialize $C\leftarrow\emptyset$, $\mathcal{Q}\leftarrow\emptyset$. Let $v_1=r,v_2,\ldots,v_n$ be the nodes of G in order of increasing distance from the root. Let $G_j=(V_j,E_j)$ the subgraph of G induced by $V_j:=\{v_1,\ldots,v_j\}$.

S2. For all $j,\ell=1,\ldots,n$, do the following. Use part (i) of Corollary 3.3 with input graph G_j and target ℓ to compute a rooted tree $Q_{j\ell}$ or rooted bipoint tree $(a_{j\ell},Q^1_{j\ell},b_{j\ell},Q^2_{j\ell})$. In the former case, add the point $\big(|V(Q_{j\ell})|,\frac{2c(Q_{j\ell})}{k}+2c_{rv_j}\big)$ to C, and add the tree $Q_{j\ell}$ to $\mathcal{Q}$. In the latter case, add the points $\big(|V(Q^1_{j\ell})|,\frac{2c(Q^1_{j\ell})}{k}+2c_{rv_j}\big)$, $\big(|V(Q^2_{j\ell})|,\frac{2c(Q^2_{j\ell})}{k}+2c_{rv_j}\big)$ to C, and add the trees $Q^1_{j\ell}$, $Q^2_{j\ell}$ to $\mathcal{Q}$.

S3. For all $\ell=1,\ldots,n$, compute $s_\ell=f(\ell)$, where $f:[1,n]\mapsto\mathbb{R}_+$ is the lower-envelope curve of C. We show in Lemma 6.5 that for every corner point $\big(\ell,f(\ell)\big)$ of f, there is some tree $Q^*_\ell\in\mathcal{Q}$ and some index j^*_ℓ such that $\ell=|V(Q^*_\ell)|$, $f(\ell)=\frac{2c(Q^*_\ell)}{k}+2c_{rv_{j^*_\ell}}$, and $\max_{v\in Q^*_\ell}c_{rv}\le c_{rv_{j^*_\ell}}$.

S4. Find a shortest $1\rightsquigarrow n$ path P_C in the concatenation graph $CG(s_1,\ldots,s_n)$.

S5. For every node $\ell>1$ on P_C, do the following. Double and shortcut Q^*_ℓ to obtain a cycle. Break this cycle into k segments, each of length at most $2c(Q^*_\ell)/k$ and add edges connecting the first and last vertex of each segment to r. This yields a collection of k cycles; traverse each resulting cycle in a random direction to obtain a collection of k tours $Z_{1,\ell},\ldots,Z_{k,\ell}$.

S6. For every $i=1,\ldots,k$, concatenate the tours $Z_{i,\ell}$ for nodes ℓ on P_C to obtain vehicle i's route.

Lemma 6.5. *(i) $s_\ell\le 4\cdot\mathsf{BNS}(k,\ell)$ for all $\ell=1,\ldots,n$. (ii) If $\big(\ell,f(\ell)\big)$ is a corner point of f, then there is a tree Q^*_ℓ and index j^*_ℓ satisfying the properties stated in step S3.*

Proof. For part (i), suppose that v_j is the node furthest from r that is covered by some optimal (k,ℓ)-bottleneck-stroll solution, so $c_{rv_j} \leq \mathsf{BNS}(k,\ell)$. Then, given part (i) of Corollary 3.3, in iteration (j,ℓ) of step S2, we add one or two points to C such that the point $\big(\ell, \frac{2z}{k} + 2c_{rv_j}\big)$, for some $z \leq k \cdot \mathsf{BNS}(k,\ell)$, lies in the convex hull of the points added. Therefore, $s_\ell = f(\ell) \leq 4 \cdot \mathsf{BNS}(k,\ell)$ since f is the lower-envelope curve of C.

The proof of part (ii) is essentially identical to the proof of Lemma 6.3 (ii). ■

Theorem 6.6. *Algorithm 3 returns a solution of cost at most* $2\mu^* \cdot \mathsf{BNSLB}$. *Hence, it is a* $2\mu^*$*-approximation algorithm for* k-MLP.

Proof. We claim that the solution returned has cost at most the length of P_C in the concatenation graph $CG(s_1,\ldots,s_n)$. Combining this with Lemma 6.5 and Theorem 2.1, we obtain that the total latency is at most $\frac{\mu^*}{2}\sum_{\ell=1}^{n} s_\ell \leq 2\mu^* \cdot \mathsf{BNSLB}$, where $\mu^* < 3.5912$.

Consider an edge (o,ℓ) of P_C. By Theorem 2.1, $\big(\ell, f(\ell)\big)$ is a corner point of f, so there exist Q^*_ℓ, j^*_ℓ satisfying the properties stated in step S3. It follows that when we transform Q^*_ℓ into k cycles, each cycle has length at most $f(\ell) = s_\ell$. Given this, the rest of the proof proceeds identically as that of Theorem 6.1. ■

7 Extensions

We now consider some extensions of multi-depot k-MLP and showcase the versatility of our algorithms by showing that our guarantees extend mostly with little effort to these problems.

Theorem 7.1. *For any* $\varepsilon > 0$*, we can compute a* $(\rho+\varepsilon)$*-approximation for the following generalizations of multi-depot* k-MLP *in time* $\mathrm{poly}\big(\text{input size}, \frac{1}{\varepsilon}\big)$*: (i) weighted sum of node latencies:* $\rho = 8.4965$*; (ii) node-depot service constraints:* $\rho = 8.4965$*; and (iii) node service times:* $\rho = 8.9965$*. The approximation ratios for (i) and (iii) improve to* $(7.1824+\varepsilon)$ *for the single-depot version.*

In some of the settings below, we will only be able to ensure that our certifiable upper bound T on the maximum latency of a node is such that $\log \mathsf{T}$ (as opposed to T) is polynomially bounded. This means that the resulting extension of (LP$_{\mathcal{P}}$) may have exponentially many variables *and* constraints. To circumvent this difficulty, we sketch below an approach for efficiently computing a $(1+\epsilon)$-approximate solution to (LP$_{\mathcal{P}}$) that only relies on $\log \mathsf{T}$ being polynomially bounded, with a $(1+\epsilon)$-violation in some constraints in the same sense as in Lemma 4.1: namely, for each i and any time point t under consideration, we use r_i-rooted *trees* of length $(1+\epsilon)t$ and total fractional weight at most $(1+\epsilon)$ (instead of a collection of r_i-rooted paths of length t of total fractional weight at most 1) to cover nodes to the extent they are covered by time t. We call such a solution a *multicriteria* $(1+\epsilon)$*-approximate solution*. This approach easily extends to solve the various LPs encountered below.

Solving (LP$_{\mathcal{P}}$) when $\log \mathsf{T}$ is polynomially bounded. Borrowing an idea from [10], we move to a compact version of (LP$_{\mathcal{P}}$) where we only have variables $\{x^i_{v,t}\}$, $\{z^i_{P,t}\}$, and constraints (2), (3) for ts in a polynomially-bounded set TS. We set $\mathsf{TS} := \{\mathsf{T}_0,\ldots,\mathsf{T}_D\}$, where $\mathsf{T}_j = \lceil (1+\epsilon)^j \rceil$, and $D = O(\log \mathsf{T}) = \mathrm{poly}(\text{input size})$ is the smallest integer such that $\mathsf{T}_D \geq \mathsf{T}$. We use (LP$^{\mathsf{TS}}_{\mathcal{P}}$) to denote this LP. The "tree-version" of (LP$^{\mathsf{TS}}_{\mathcal{P}}$) is obtained similarly from (LP$_{\mathcal{T}}$) and denoted (LP$^{\mathsf{TS}}_{\mathcal{T}}$).

Define $\mathsf{T}_{-1} = 0$ Given a solution (x,z) to (LP$_{\mathcal{P}}$), where t ranges from 1 to T, we can define (x',z') as follows: set $z'^i_{P,t} = z^i_{P,t}$ for all i, $P \in \mathcal{P}_t$ and $t \in \mathsf{TS}$; set $x'^i_{v,T_j} = \sum_{t\in T_{j-1}+1}^{T_j} x^i_{v,t}$ for all i, v, and $T_j \in \mathsf{TS}$. It is not hard to see that (x',z') is feasible to (LP$^{\mathsf{TS}}_{\mathcal{P}}$) and that its cost is at most $(1+\epsilon)$ times the cost of (x,z). Thus, the optimal value of (LP$^{\mathsf{TS}}_{\mathcal{P}}$) is at most $(1+\epsilon)\,OPT_{\mathcal{P}}$. Conversely, given a solution (x',z') to (LP$^{\mathsf{TS}}_{\mathcal{P}}$), setting $x^i_{v,t}$ equal to $x'^i_{v,t}$ if $t \in \mathsf{TS}$ and 0 otherwise, and $z^i_{P,t} = z'^i_{P,\mathsf{T}_j}$ for all $t \in [\mathsf{T}_j, \mathsf{T}_{j+1})$ and all j, yields a feasible solution to (LP$_{\mathcal{P}}$) of the same cost.

Since $(\text{LP}_{\mathcal{P}}^{\mathsf{TS}})$ is an LP of the same form as $(\text{LP}_{\mathcal{P}})$ but with polynomially many variables, we can approximately solve it in the sense of Lemma 4.1: for any $\epsilon > 0$, we can obtain in time $\text{poly}\big(\text{input size}, \frac{1}{\epsilon}\big)$ a solution to $(\text{LP}_{\mathcal{T}}^{\mathsf{TS}})$ of cost at most $OPT_{(\text{LP}_{\mathcal{P}}^{\mathsf{TS}})} \leq (1+\epsilon)OPT_{\mathcal{P}}$ with a $(1+\epsilon)$-violation in some constraints. This in turn yields a solution to $(\text{LP}_{\mathcal{P}})$ of no greater cost and with the same $(1+\epsilon)$-violation in some constraints.

Observe that the above idea of restricting time points to the polynomially-bounded set $\{\mathsf{T}_0, \ldots, \mathsf{T}_D\}$ also applies to (LP3) and shows that we can obtain a feasible solution to (LP3) of cost at most $(1+\epsilon)\,OPT_{\text{LP3}}$ in time $\text{poly}\big(\text{input size}, \frac{1}{\epsilon}\big)$ while only assuming that $\log \mathsf{T} = \text{poly}(\text{input size})$.

7.1 Weighted sum of node latencies

Here, we have nonnegative node weights $\{w_v\}$ and want to minimize the weighted sum $\sum_v w_v(\text{latency of } v)$ of node latencies. We again have the upper bound $\mathsf{T} = 2n\mathsf{LB}$ on the maximum latency of a node. We cannot use scaling and rounding to ensure that $\mathsf{T} = \text{poly}(\text{input size})$, but note that $\log \mathsf{T} = \text{poly}(\text{input size})$.

For multi-depot k-MLP, we consider $(\text{LP}_{\mathcal{P}})$ with the objective modified to take into account the node weights. We can obtain a multicriteria $(1+\epsilon)$-approximate solution to the resulting LP as described above. We round this as before; this works since Lemma 5.4 remains unchanged and bounds the expected latency of each node in terms of the latency it incurs under the LP solution. The only minor change is that in the truncated version (Remark 5.5), we set $N = D + \kappa \ln\big(\frac{(\max_v w_v)n\mathsf{T}}{\varepsilon}\big)$ since covering an uncovered node at the end incurs weighted latency at most $(\max_v w_v)\mathsf{T}$.

A 7.183-approximation for k-MLP. Both the LP-rounding and the combinatorial algorithms in Section 6 can be extended to this setting. We describe the LP-rounding algorithm here; the extension of the combinatorial algorithm is descibed in Appendix D. We consider (LP3) with the weighted-latency objective and obtain a $(1+\epsilon)$-approximate solution (x, z) to this LP. We round this losing a $2\mu^*$-factor in a very similar fashion to Algorithm 2 in Section 6.1. We may assume via scaling that all weights are integers, and $w_r = 1$. Let $W = \sum_{u \in V} w_u$. A naive extension of the algorithm in Section 6 would be to create w_v nodes co-located at v and include a node in the concatenation graph for every possible weight value from 0 to $\sum_v w_v$. But this only yields pseudopolynomial running time. Instead, we proceed as follows.

Let $\mathsf{TS} := \{\mathsf{T}_0, \ldots, \mathsf{T}_D\}$ be the time points that we consider when solving (LP3) approximately, where $\mathsf{T}_j = \lceil (1+\epsilon)^j \rceil$ for all $j \geq 0$, and $D = O(\log \mathsf{T}) = \text{poly}(\text{input size})$ is the smallest integer such that $\mathsf{T}_D \geq \mathsf{T}$. In Algorithm 2, we always only consider time points in TS. Step R1 is unchanged. In step R2, for each time $t \in \mathsf{TS}$ and each arborescence Q_ℓ^t of the weighted arborescence family obtained for time t we now add the point $\big(w(V(Q_\ell^t) \cap S(t)), \frac{2c(Q_\ell^t)}{k} + 2t\big)$ to C, and as before, add Q_ℓ^t to $\mathcal{Q}$.

Let $f : [1, W] \mapsto \mathbb{R}_+$ be the lower-envelope curve of C. We claim that the shortest path P_C in the concatenation graph $CG\big(f(1), \ldots, f(W)\big)$ can be computed efficiently. This is because by Theorem 2.1, the shortest path only uses nodes corresponding to corner points of f. So the shortest path remains unchanged if we only consider edges in the concatenation graph incident to such nodes. This subgraph of the concatenation graph has polynomial size (and can be computed) since all corner points of f must be in C and $|C| = O(D)$. Moreover, as in part (ii) of Lemma 6.3, every corner point $\big(\ell, f(\ell)\big)$ of f corresponds to some tree $Q_\ell^* \in \mathcal{Q}$ and some $t^*\ell \in \mathsf{TS}$ such that $\ell = w\big(V(Q_\ell^*) \cap S(t_\ell^*)\big)$, $f(\ell) = \frac{2c(Q_\ell^*)}{k} + 2t_\ell^*$, and $\max_{v \in Q_\ell^* \cap S(t_\ell^*)} c_{rv} \leq t_\ell^*$. Given this, steps R5, R6 are unchanged.

The analysis also proceeds as before. Mimicking the proof of Theorem 6.1, we can again argue that the solution returned has cost at most the length of P_C in $CG\big(f(1), \ldots, f(W)\big)$. To complete the analysis, utilizing Corollary 2.2, we need to bound $\int_1^W f(x)dx$. Recall that $x'_{v,t} = \sum_i x_{v,t}$ for all $v \neq r, t \in \mathsf{TS}$. As before, define $x'_{r,0} = 1$, $x'_{r,t} = 0$ for all $t > 0$. Also, define $x'_{v,t} = 0$ for all $v \neq r$ and all $t < \mathsf{T}_D, t \notin \mathsf{TS}$. Generalizing part (i) of Lemma 6.3, we show that $\int_1^W f(x)dx$ is at most $4 \sum_{u \in V, t \in \mathsf{TS}} w_u \cdot t x'_{u,t}$.

Define $\mathsf{T}_{-1} := 0$. Dovetailing the proof of Lemma 6.2, we have that $\big(\int_1^W f(x)dx\big)/4 = \int_0^{f(W)/4}\big(W - f^{-1}(4y)\big)dy$. Note that $f(W) \le 4\mathsf{T}_D$. For any $t = \mathsf{T}_j \in \mathsf{TS}$, we include all points generated by arborescences in the weighted arborescence family for t in C. So we ensure that some point (a, b), where $a \ge \sum_{u\in V}\sum_{t'=0}^{t} w_u x'_{u,t'}$, $b \le 4t$ lies in the convex hull of C. So $f^{-1}(4t) \ge \sum_{u\in V}\sum_{t'=0}^{t} w_u x'_{u,t'}$; this also holds for $t = 0$. So as in the proof of Lemma 6.2, we have

$$\int_0^{f(W)/4}\big(W - f^{-1}(4y)\big)dy \le \sum_{j=0}^{D}(\mathsf{T}_j - \mathsf{T}_{j-1})\big(W - f^{-1}(4\mathsf{T}_{j-1})\big) = \sum_{j=0}^{D}(\mathsf{T}_j - \mathsf{T}_{j-1})\big(W - \sum_{u\in V}\sum_{t'=0}^{\mathsf{T}_{j-1}} w_u x'_{u,t'}\big)$$
$$= \sum_{j=0}^{D}(\mathsf{T}_j - \mathsf{T}_{j-1})\sum_{u\in V} w_u \sum_{t'\ge \mathsf{T}_j} x'_{u,t'} = \sum_{u\in V}\sum_{t'\in\mathsf{TS}} w_u \cdot t' x'_{u,t'}$$

7.2 Node-depot service constraints

In this setting, we are given a set $S_v \subseteq R$ of depots for each node v, and v must be served by a vehicle originating at a depot in S_v. The 8.497-approximation algorithm extends in a straightforward manner. We now define $\mathsf{LB} := \max_v \min_{i\in S_v} c_{r_i v}$, and can again ensure that LB, and hence $\mathsf{T} = 2n\mathsf{LB}$ is polynomially bounded. We modify constraint (1) of (LP$_{\mathcal{P}}$) to $\sum_{t,i\in S_v} x^i_{v,t} \ge 1$, obtain a solution to the resulting LP via Lemma 4.1, and round it as before.

7.3 Node service times

Here, each non-root node v has a service time d_v that is added to the latency of node v, and every node visited after v on the path of the vehicle serving v. Set $d_r = 0$ for $r \in R$ for notational convenience. We can set $\mathsf{T} = \sum_v d_v + 2n\mathsf{LB}$ as an upper bound on the maximum latency of a node.

Let $c''_{uv} = c_{uv} + \frac{d_u+d_v}{2}$ for all u, v. Observe that the c''_es form a metric. We obtain a multicriteria $(1+\epsilon)$-approximate solution (x, z) to (LP$_{\mathcal{P}}$) with the c''-metric. Note that this LP is a valid relaxation since if P is the portion of a vehicle's route up to and including node v then $c''(P)$ is at most the latency of v. We round (x, z) as in Algorithm 1. The additive 0.5 increase in approximation comes from the fact that when we convert a tree Q of c''-cost t into a cycle Z, the expected contribution to the latency of a node $v \in Z$ is now at most $d_v + \frac{1}{2}\big(2c''(Q) - d_v\big) \le t + \frac{d_v}{2}$. Thus, we obtain an 8.997-approximation.

A 7.183-approximation for k-MLP. Define the *mixed length* of a path or tree Q to be $c(Q) + d(V(Q))$. Defining the *directed* metric $c'_{u,v} = c_{uv} + d_v$ for all u, v, note that if we have a rooted tree and we direct its edges away from r, then its c'-cost is exactly its mixed length (since $d_r = 0$). Again, both the LP-rounding and combinatorial algorithms in Section 6 extend with small changes. Essentially, the change is that we work with the c'-metric, which works out in the LP-rounding algorithm since Theorem 3.1 does not depend in any way on the edge costs, and works out in the combinatorial algorithm since Theorems 3.2 and Corollary 3.3 also apply with the c'-metric and yield analogous statements where the c-cost is replaced by the mixed-length objective. The only thing to verify is that the procedure for converting a tree Q of mixed length (i.e., c'-cost) at most kt into k tours ensures that the expected contribution to the latency of a node $v \in Q$ is at most t. We describe the LP-rounding algorithm here and the combinatorial algorithm in Appendix E.

Let (x, z) be a $(1+\epsilon)$-approximate solution (x, z) to (LP3) with arc-costs $\{c'_a\}_{a\in A}$ (instead of the c-metric), obtained by considering time points in $\mathsf{TS} := \{\mathsf{T}_0, \ldots, \mathsf{T}_D\}$. Here $\mathsf{T}_j = \lceil(1+\epsilon)^j\rceil$ for all $j \ge 0$, and $D = O(\log \mathsf{T}) = \text{poly}(\text{input size})$ is the smallest integer such that $\mathsf{T}_D \ge \mathsf{T}$. As before, let $x'_{v,t} = \sum_i x^i_{v,t}$ and $z'_{a,t} = \sum_i z^i_{a,t}$. Define $x'_{r,0} = 1$, $x'_{r,t} = 0$ for all $t > 0$, and $x'_{v,0} = 0$ for all $v \ne r$. In Algorithm 2, we always only consider time points in TS. Steps R1, R4 are unchanged. The only change in

steps R2, R3 is that the c-cost is replaced by the mixed-length objective (i.e., the c'-cost of the out-tree rooted at r).

The main change is in step R5, where we need to be more careful in obtaining the collection of k tours $Z_{1,\ell}, \ldots, Z_{k,\ell}$ that cover $V(Q^*_\ell) \cap S(t^*_\ell)$ from the rooted tree Q^*_ℓ. We show that we can obtain these k tours so that we have

$$c(Z_{i,\ell}) + 2d(V(Z_{i,\ell})) \leq 2 \cdot \frac{c(Q^*_\ell) + d\big(V(Q^*_\ell)\big)}{k} + 2t^*_\ell \qquad \text{for all } i = 1, \ldots, k.$$

This follows from Lemma 7.2, where we prove that we can obtain k cycles satisfying the above inequality; traversing each cycle in a random direction, clockwise or counterclockwise, yields the desired k tours.

Lemma 7.2. *Let Q be a rooted tree. Let $S \subseteq V(Q)$ and $L = \max_{u \in S}(c_{ru} + d_u)$. We can obtain k cycles $Z_1, \ldots, Z_k$ that together cover S such that $c(Z_i) + 2d(V(Z_i)) \leq 2\frac{c(Q)+d(V(Q))}{k} + 2L$ for all $i = 1, \ldots, k$.*

Proof. Assume that S contains a node other than the root r, otherwise, we can take $Z_1, \ldots, Z_k$ to be the trivial ones consisting of only r. Let $M = \frac{c(Q)+d(V(Q))}{k}$. Pick an arbitrary node $w \in S$, $w \neq r$. First, we double and shortcut Q to remove nodes not in S and repeat occurrences of nodes, to obtain an r-w path P satisfying $c(P) \leq 2c(Q)$, and therefore, $c(P) + 2d(v(P)) \leq 2Mk$. Note that $c_{rv} + d_v \leq L$ for every $v \in P$. We obtain the cycles by snipping P at appropriate places and joining the resulting segments to the root. To bound the cost of each resulting cycle and argue that the snipping process creates at most k segments, we define two charges charge^1 and charge^2 for each segment that, roughly speaking, sandwich the quantity of interest $c(.) + 2d(.)$ for each segment. For $u, v \in P$, let P_{uv} denote the portion of P between (and including) nodes u and v.

Set $u_1 = r$. We repeatedly do the following. Let v be the first node after u_i on P such that one of the following holds. If there is no such node, then we terminate the loop, and set $v_i = w$.

(i) $c(P_{u_iv}) + 2d(V(P_{u_iv})) - d_{u_i} - d_v > 2M$. Note that in this case P_{u_iv} consists of at least two edges, since otherwise we have

$$c(P_{u_iv}) + 2d(V(P_{u_iv})) - d_{u_i} - d_v = c_{u_iv} + d_{u_i} + d_v \leq c_{ru_i} + d_{u_i} + c_{rv} + d_v \leq 2L.$$

We set set $u_{i+1} = v$, and v_i to be the node immediately before v on Z. Define $\mathsf{charge}^1_i = c(P_{u_iv_i}) + 2d(V(P_{u_iv_i})) - d_{u_i} - d_{v_i}$ and $\mathsf{charge}^2_i = c(P_{u_iv}) + 2d(V(P_{u_iv})) - d_{u_i} - d_v$. We say that $P_{u_iv_i}$ is of type (i).

(ii) $c(P_{u_iv}) + 2d(V(P_{u_iv})) - d_{u_i} - d_v \leq 2M < c(P_{u_iv}) + 2d(V(P_{u_iv})) - d_{u_i}$. We set $v_i = v$. Define $\mathsf{charge}^1_i = c(P_{u_iv}) + 2d(V(P_{u_iv})) - d_{u_i} - d_v$ and $\mathsf{charge}^2_i = c(P_{u_iv}) + 2d(V(P_{u_iv})) - d_{u_i}$. We say that $P_{u_iv_i}$ is of type (ii). If $v = w$, then we terminate the loop; otherwise, we set u_{i+1} to be the node immediately after v on P.

We increment i and repeat the above process.

Suppose we create q segments in the above process, i.e., q is the value of the counter i at termination. We first argue that $q \leq k$. To see this, note that charge^2_i is certainly well defined for all $i = 1, \ldots, q-1$, and by definition, $\mathsf{charge}^2_i > 2M$ for all $i = 1, \ldots, q-1$. So $\sum_{i=1}^{q-1} \mathsf{charge}^2_i > 2M(q-1)$. We claim that $\sum_{i=1}^{q-1} \mathsf{charge}^2_i \leq c(P) + 2d(V(P)) \leq 2Mk$, and therefore $q - 1 < k$. To see the claim, notice that every edge of P contributes to at most one charge^2_i. Also, every node v contributes in total at most $2d_v$ to $\sum_{i=1}^{q-1} \mathsf{charge}^2_i$. This is certainly true if $v \notin \{u_1, \ldots, u_q\}$; otherwise if $v = u_i$, then it contributes d_v to charge^2_i, and possibly d_v to charge^2_{i-1}, if $P_{u_{i-1}v_{i-1}}$ is of type (i).

Each segment $P_{u_iv_i}$ yields a cycle Z_i by joining u_i and v_i to r; the remaining $k - q$ cycles are the trivial ones consisting of only $\{r\}$. If charge^1_q has not been defined (which could happen if no node v satisfies

(i) or (ii) when $i = q$), define it to be $c(P_{u_q v_q}) + 2d(V(P_{u_q v_q})) - d_{u_q} - d_{v_q}$. For $i = 1, \ldots, q$, note that by definition, we have $c(P_{u_i v_i}) + 2d(V(P_{u_i v_i})) - d_{u_i} - d_{v_i} = \mathsf{charge}_i^1 \leq 2M$. So $c(Z_i) + 2d(V(Z_i)) = (c_{ru_i} + d_{u_i}) + (c_{rv_i} + d_{v_i}) + \mathsf{charge}_i^1 \leq 2M + 2t$. ■

The remainder of the analysis dovetails the one in Section 6.1. Analogous to Lemma 6.3 (and as in the weighted-latency setting), we have that the lower-envelope curve f satisfies: (i) $\int_1^n f(x)dx \leq 4\sum_{u \in V, t \in \mathsf{TS}} t x'_{u,t}$, and and (ii) every corner point $\big(\ell, f(\ell)\big)$ satisfies the properties stated in the modified step R3. Finally, we argue that the cost of the solution returned is at most the length of the shortest path P_C in $CG\big(f(1), \ldots, f(n)\big)$, which yields an approximation guarantee of $2\mu^*(1+\epsilon)$ (where $\mu^* < 3.5912$).

As before, consider an edge (o, ℓ) of P_C. Assume inductively that we have covered at least o nodes by the partial solution obtained by concatenating tours corresponding to the portion of P_c up to and including o. By Lemma 7.2, and since $\big(\ell, f(\ell)\big)$ is a corner point of f, each $Z_{i,\ell}$ has mixed length at most $f(\ell)$. Also, concatenating $Z_{i,\ell}$ to vehicle i's route, for all $i = 1, \ldots, k$, we end up covering at least ℓ nodes. The increase in latency due to this step for a node that remains uncovered after this step is at most $f(\ell)$. Consider a node v that is covered in this step, and say $v \in Z_{i,\ell}$. Since $Z_{i,\ell}$ is obtained by traversing the corresponding cycle in a random direction, the increase in latency of v is at most

$$d_v + \frac{1}{2} \cdot \Big(c(Z_{i,\ell}) + d\big(V(Z_{i,\ell})\big) - d_v \Big) \leq \frac{1}{2} \cdot \Big(c(Z_{i,\ell}) + d\big(V(Z_{i,\ell})\big) + d_v \Big) \leq \frac{c(Z_{i,\ell}) + 2d\big(V(Z_{i,\ell})\big)}{2} \leq \frac{f(\ell)}{2}.$$

The last inequality follows from Lemma 7.2. Therefore, as before, the increase in total latency due to this step is at most the length of the (o, ℓ) edge in $CG\big(f(1), \ldots, f(n)\big)$. So by induction, the total latency is at most the length of P_c in $CG\big(f(1), \ldots, f(n)\big)$.

8 Proof of Theorem 4.2

Part (iii) is simply a restatement of Lemma 4.1, so we focus on parts (i) and (ii).

Proof of part (i). The proof follows from some simple algebraic manipulations. We express both the objective value of (LP2$_\mathcal{P}$) and BNSLB equivalently as the sum over all time units t of the number of uncovered nodes at time t (i.e., after time $t-1$), where for the LP, by "number" we mean the total extent to which nodes are not covered. We then observe that this "number" for an LP solution is at least the corresponding value in the expression for BNSLB.

Let $b^*_\ell = \mathsf{BNS}(k, \ell)$ for $\ell = 1, \ldots, n$. Note that b^*_ℓ is an integer for all ℓ since all c_es are integers, and $b^*_\ell = 0$ for all $\ell \leq |R|$. Let (x, z) be an optimal solution to (LP2$_\mathcal{P}$). For convenience, we set $x_{v,t} = 0 = z_{\vec{P},t}$ for all $t > \mathsf{T}$ and all $v, \vec{P} \in \mathcal{P}_t$. Also set $x_{r_i,t} = 0$ for all $t \geq 1$. Let u index nodes in V. Define $n^*_t = \max\{\ell : b^*_\ell \leq t\}$ for all $t \geq 1$, and $N_t = \sum_{u, t' \leq t} x_{u,t'}$ for all $t \geq 1$. Note that $N_t \leq \sum_u \sum_{\vec{P} \in \mathcal{P}_t : u \in \vec{P}} z_{\vec{P},t} = \sum_{\vec{P} \in \mathcal{P}_t} z_{\vec{P},t} |\{u : u \in \vec{P}\}| \leq n^*_t$ for all t.

We now express both the objective value of (LP2$_\mathcal{P}$) and BNSLB equivalently as the sum over all time units t of the number of uncovered nodes at time t (i.e., after time $t-1$). This coupled with the fact that $N_t \leq n^*_t$ for all t completes the proof. We have

$$\mathsf{BNSLB} = \sum_{\ell=1}^{n} b^*_\ell = \sum_{\ell=1}^{n} \sum_{t=1}^{b^*_\ell} 1 = \sum_{t=1}^{b^*_n} \sum_{\ell : b^*_\ell \geq t} 1 = \sum_{t=1}^{b^*_n} (n - n^*_{t-1}) = \sum_{t \geq 1} (n - n^*_{t-1}).$$

We also have

$$OPT_{\text{LP2}_{\mathcal{P}}} = \sum_{t\geq 1,u} t x_{u,t} = \sum_{t} \Big(\sum_{t'=1}^{t} 1\Big) \sum_{u} x_{u,t}$$
$$= \sum_{t'\geq 1} \sum_{t\geq t',u} x_{u,t} \geq \sum_{t'\geq 1} (n - N_{t'-1}) \geq \sum_{t'\geq 1} (n - n^*_{t'-1}) \geq \mathsf{BNSLB}.$$

Proof of part (ii). We prove the second statement, which immediately implies the first. Let $\kappa = 1 + \epsilon$. Let (x, z) be a solution to $\big(\text{LP2}^{(\kappa)}_{\mathcal{T}}\big)$. The rounding procedure and its analysis are very similar to the one in Section 5. Let $h = c^{\Gamma}$, where $\Gamma \sim U[0, 1)$. At each time $t_j := hc^j$, for $j = 0, 1, \ldots$, we sample a tree configuration $\vec{Q} = (Q_1, \ldots, Q_k)$ from the distribution $\big\{z_{\vec{Q},t_j}/\kappa\big\}_{\vec{Q}\in\mathcal{T}_{\kappa t}}$; we convert each Q_i into a cycle and traverse this cycle in a random direction to obtain a tour $Z_{i,j}$. We then concatenate the tours $Z_{i,0}, Z_{i,1}, \ldots$ for all $i = 1 \ldots, k$.

Define t_{-1}, Δ_j, lat_v, $p_{v,j}$, L_v for all v, j as in the analysis in Section 5. Define $y_{v,t} := \sum_{t'\leq t} x_{v,t'}$ and $o'_{v,j} := 1 - y_{v,t_j}$, for all v, t, j; let $o'_{v,j} = 1$ for all v and $j < 0$. Let $\mathsf{lat}'_v := \sum_{j\geq 0} o'_{v,j-1}\Delta_j$. Let $\mathrm{E}^h[.]$ and $\mathrm{E}[.]$ denote the same quantities as in the analysis in Section 5. Parts (ii) and (iii) of Claim 5.2 continue to hold. They key difference is that we obtain an improved expression for $p_{v,j}$ compared to the one in Lemma 5.3. We now have that $p_{v,j}$ is almost $o'_{v,j}$ since instead of sampling k trees independently as in Algorithm 1 (which incurs a loss since $\prod_i(1 - a_i)$ is smaller than $\sum_i a_i$), we now sample a *single tree configuration*; this improved bound also results in the improved approximation. More precisely, mimicking the proof of Lemma 5.3, we obtain that $p_{v,j} \leq \frac{o'_{v,j}}{\kappa} + \big(1 - \frac{1}{\kappa}\big)p_{v,j-1}$ for all v and $j \geq -1$. Plugging this in the proof of Lemma 5.4 gives $\mathrm{E}[L_v] \leq \frac{c+1}{(\ln c)(1-c(1-1/\kappa))} \cdot \mathsf{lat}_v \leq \frac{c+1}{(\ln c)(1-c\epsilon)} \cdot \mathsf{lat}_v$ for all v.

The expression $\frac{c+1}{\ln c}$ achieves its minimum value of μ^* at $c = \mu^*$ (i.e., when $c + 1 = c \ln c$), so the approximation factor is at most $\frac{\mu^*}{1-\mu^*\epsilon}$.

References

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

[2] A. Archer and A. Blasiak. Improved approximation algorithms for the minimum latency problem via prize-collecting strolls. In *Proceedings of the 21st SODA*, pages 429–447, 2010.

[3] A. Archer, A. Levin, and D. Williamson. A faster, better approximation algorithm for the minimum latency problem. *SIAM J. Comput.*, 37(5):1472–1498, 2008.

[4] S. Arora and G. Karakostas. Approximation schemes for minimum latency problems. *SIAM Journal on Computing*, 32(5):1317–1337, 2003.

[5] G. Ausiello, S. Leonardi, and A. MarchettiSpaccamela. On salesmen, repairmen, spiders and other traveling agents. In *Proceedings of the Italian Conference on Algorithms and Complexity*, pages 1–16, 2000.

[6] J. Bang-Jensen, A. Frank, and B. Jackson. Preserving and increasing local edge-connectivity in mixed graphs. *SIAM Journal on Discrete Math.*, 8(2):155–178, 1995.

[7] L. Bianco, A. Mingozzi, and S. Ricciardelli. The traveling salesman problem with cumulative costs. *Networks*, 23(2):81–91, 1993.

[8] A. Blum, P. Chalasani, D. Coppersmith, B. Pulleyblank, P. Raghavan, and M. Sudan. The Minimum Latency Problem. In *Proceedings of 26th STOC*, pages 163–171, 1994.

[9] G. Calinescu, C. Chekuri, M. Pál, and J. Vondrák. Maximizing a monotone submodular function subject to a matroid constraint. *SIAM J. Comput.*, 40(6):1740–1766, 2011.

[10] D. Chakrabarty and C. Swamy. Facility location with client latencies: linear-programming based techniques for minimum latency problems. In *Proceedings of the 15th IPCO*, pages 92–103, 2011.

[11] K. Chaudhuri, P. B. Godfrey, S. Rao, and K. Talwar. Paths, Trees and Minimum Latency Tours. In *Proceedings of 44th FOCS*, pages 36–45, 2003.

[12] C. Chekuri and A. Kumar. Maximum coverage problem with group budget constraints and applications. In *Proceedings of the 17th APPROX*, pages 72–83, 2004.

[13] J. Fakcharoenphol, C. Harrelson, and S. Rao The k-traveling repairman problem. In *Proceedings of the 14th SODA*, pages 655–664, 2003.

[14] J. Fakcharoenphol, C. Harrelson, and S. Rao The k-traveling repairman problem. *ACM Trans. on Alg.*, Vol 3, Issue 4, Article 40, 2007.

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

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

[17] N. Garg. Saving an epsilon: a 2-approximation for the k-MST problem in graphs. In *Proceedings of the 37th STOC*, pages 396–402, 2005.

[18] M. Goemans and J. Kleinberg. An improved approximation ratio for the minimum latency problem. In *Proceedings of 7th SODA*, pages 152–158, 1996.

[19] M. Goemans and D. Williamson. A general approximation technique for constrained forest problems. *SIAM Journal on Computing*, 24:296–317, 1995.

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

[21] A. Lucena. Time-dependent traveling salesman problem – the deliveryman case. *Networks*, 20(6):753–763, 1990.

[22] E. Minieka. The delivery man problem on a tree network. *Annals of Operations Res.*, 18:261–266, 1989.

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

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

[25] R. Sitters. Polynomial time approximation schemes for the traveling repairman and other minimum latency problems. In *Proceedings of the 25th SODA*, pages 604–616, 2014.

[26] R. Sitters. The minimum latency problem is NPhard for weighted trees. In *Proceedings of the 9th Conference on Integer Programming and Combinatorial Optimization*, pages 230–239, 2002.

[27] P. Toth and D. Vigo, eds. *The Vehicle Routing Problem*. SIAM Monographs on Discrete Mathematics and Applications, Philadelphia, 2002.

A Proofs from Section 2

Proof of Corollary 2.2. We argue that the length L of the shortest $1 \rightsquigarrow n$ path in $CG\big(f(1),\dots,f(n)\big)$ is at most the claimed bound, which implies the claimed statement.

For $x \in [1, 1+k(n-1)]$, define $f_k(x) := f\big(1+\frac{x-1}{k}\big)$. Note that for any $x \in [1, 1+k(n-1)]$, $(x, f_k(x))$ is a corner point of f_k iff $(x', f(x'))$, where $x' = 1 + \frac{x-1}{k}$ is a corner point of f. Also, if $(x', f(x'))$ is a corner point of f, then x' must be an integer and $f(x') = C_{x'}$. Hence, if $(x, f_k(x))$ is a corner point of f_k, then $1 + \frac{x-1}{k}$ must be an integer.

Now consider the shortest $1 \rightsquigarrow N := 1 + k(n-1)$ path P_k in $CG\big(f_k(1), f_k(2), \dots, f_k(N)\big)$. Let L_k be the length of P_k. Consider an edge (o, ℓ) of P_k. Let $o' = 1 + \frac{o-1}{k}$, $\ell' = 1 + \frac{\ell-1}{k}$. By Theorem 2.1 and the above discussion, o' and ℓ' must be integers. The cost of (o, ℓ) is

$$\begin{aligned} f_k(\ell)\Big(N - \frac{o+\ell}{2}\Big) &= f(\ell')\Big(k(n-1) - \frac{o-1+\ell-1}{2}\Big) \\ &= k \cdot f(\ell')\Big(n-1-\frac{(o-1)/k + (\ell-1)/k}{2}\Big) = k \cdot f(\ell')\Big(n - \frac{o'+\ell'}{2}\Big) \end{aligned}$$

which is the k times the cost of the (o', ℓ') edge in $CG\big(f(1),\dots,f(n)\big)$. Thus, $L \le L_k/k$ for all $k \ge 1$.

Moreover,

$$\frac{L_k}{\mu^*/2} \le \sum_{x=1}^{N} f_k(x) \le \sum_{x=2}^{N}\Big(\int_{x-1}^{x} f_k(t)dt + f_k(x) - f_k(x-1)\Big) = \int_1^N f_k(t)dt + f_k(N) = k\int_1^n f(x)dx + f(n).$$

Therefore, $\frac{L}{\mu^*/2} \le \int_1^n f(x)dx + \frac{f(n)}{k}$ for all $k \ge 1$, which implies that $L \le \frac{\mu^*}{2}\int_1^n f(x)dx$. ∎

Proof of Theorem 2.3. Part (ii) for MLP follows from the analysis in [2], so we focus on part (i). We use a concatenation-graph argument similar to the one used for single-vehicle MLP in [18]. Let $b^*_\ell = \mathsf{BNS}(k, \ell)$ for $\ell = 1, \dots, n$. Consider any sequence $0 < \ell_1 < \ell_2 < \dots < \ell_h = n$ of indices. We can obtain a solution from this as follows. For each index $\ell = \ell_p$ and each $i = 1, \dots, k$, we double and shortcut the r_i-rooted path in the optimal solution to the (k, ℓ)-bottleneck stroll problem and traverse the resulting cycle in a random direction to obtain a tour $Z_{i,\ell}$ (which contains r_i). For every $i = 1, \dots, k$, we then concatenate the tours $Z_{i,\ell_1}, \dots, Z_{i,\ell_h}$. Since $\ell_h = n$, this covers all nodes, so we obtain a feasible solution to the multi-depot k-MLP instance.

We show that the cost of this solution is at most the length of the $1 \to \ell_1 \to \dots \to \ell_h$ path in the concatenation graph $CG(2b^*_1, \dots, 2b^*_n)$, so the statement follows from Theorem 2.1.

To bound the cost, consider an edge (o, ℓ) of the path. Suppose inductively that we have covered at least o nodes by the partial solution constructed by concatenating tours corresponding to the nodes on the path up to and including o. Consider the additional contribution to the total latency when we concatenate tour $Z_{i,\ell}$ to vehicle i's current route, for $i = 1, \dots, k$. The resulting partial solution covers at least ℓ nodes (since $\bigcup_i V(Z_{i,\ell}) \ge \ell$). Suppose that in this step we cover B additional nodes, and there are A uncovered nodes remaining after this step. Then, $B \le \ell - o$ and $A \le n - \ell$. The latency of the uncovered increases by at most $\max_i c(Z_{i,\ell}) \le 2b^*_\ell$. The latency of each node u that got covered by, say, $Z_{i,\ell}$ increases by at most

$\frac{c(Z_{i,\ell})}{2} \le b^*_\ell$ since we choose a random direction for traversing the cycle $Z_{i,\ell}$. Therefore, the total increase in latency is at most

$$2b^*_\ell\Big(A + \frac{B}{2}\Big) \le 2b^*_\ell\Big(n - \ell + \frac{\ell - o}{2}\Big) = 2b^*_\ell\Big(n - \frac{o+\ell}{2}\Big).$$

This is precisely the length of the (o, ℓ) edge in $CG(2b^*_1, \ldots, 2b^*_n)$, and so by induction, the total latency of the solution is at most the length of the $1 \to \ell_1 \to \ldots \to \ell_h$ path in $CG(2b^*_1, \ldots, 2b^*_n)$. ■

B Proof of Theorem 3.1

We restate the theorem for easy reference.

Theorem 3.1. *Let $D = (U + r, A)$ be a digraph with nonnegative integer edge weights $\{w_e\}$, where $r \notin U$ is a root node, such that $|\delta^{\text{in}}(u)| \ge |\delta^{\text{out}}(u)|$ for all $u \in U$. For any integer $K \ge 0$, one can find out-arborescences $F_1, \ldots, F_q$ rooted at r and integer weights $\gamma_1, \ldots, \gamma_q$ in polynomial time such that $\sum_{i=1}^q \gamma_i = K$, $\sum_{i:e\in F_i} \gamma_i \le w_e$ for all $e \in A$, and $\sum_{i:u\in F_i} \gamma_i \ge \min\{K, \lambda_D(r, u)\}$ for all $u \in U$.*

We require some notation and lemmas proved by [6]. To avoid confusion we use the superscript * when referring to statements in [6]. Theorem 3.1 is a polytime version of Corollary* 2.1 in [6], and our proof closely follows that of Theorem* 2.6 in Bang-Jensen et al. [6]. Let $\Lambda_D(u, v) = \min\{K, \lambda_D(u, v)\}$ be the required connectivity.

Definition B.1. Let $e = (t, u)$ and $f = (u, v)$ be edges. *Splitting off* e and f means removing e and f and adding a new edge (t, v), or, in a weighted graph, subtracting some amount $x > 0$ from the weights w_e and w_f and increasing $w_{(t,v)}$ by x. We denote the new digraph by D^{ef}. Edges e and f are *splittable* if $\lambda_{D^{ef}}(x, y) \ge \Lambda_D(x, y)$ for all $x, y \ne u$.

Say that u and v are *separated* by X is $|X \cap \{u, v\}| = 1 = |\{u, v\} \setminus X|$. We call a set of nodes X *tight* if $\min\{|\delta^{\text{in}}(X)|, |\delta^{\text{out}}(X)|\} = \max_{u,v \text{ separated by } X} \Lambda_D(u, v)$, that is, X is a minimum cut for some u, v maximum flow, and we say that X is tight for u, v if u, v are separated by X and $\min\{|\delta^{\text{in}}(X)|, |\delta^{\text{out}}(X)|\} = \Lambda_D(u, v)$. If $t, v \in X$ and $u \notin X$, splitting edges $(t, u), (u, v)$ reduces $|\delta^{\text{in}}(X)|$ and $|\delta^{\text{out}}(X)|$, so there is a close relationship between splittable edges and tight sets. Note that for an Eulerian digraph $D = (V, A)$ (i.e., $|\delta^{\text{in}}(u)| = |\delta^{\text{out}}(u)|$ for all $u \in V$), we have $|\delta^{\text{in}}(X)| = |\delta^{\text{out}}(X)|$ for all $X \subseteq V$, and $\Lambda_D(u, v) = \Lambda_D(v, u)$ for all $u, v \in V$.

Lemma B.2 (Claim* 2.1 in [6])**.** *Edges $e = (t, u)$ and $f = (u, v)$ are splittable if and only if there is no set X such that $t, v \in X$, $u \notin X$, and X is tight for some $x, y \ne u$.*

Lemma B.2 can be used to prove that splittable edges always exist.

Lemma B.3 (Theorem* 2.2 in [6])**.** *Let $D = (V, A)$ be an Eulerian digraph and $v \in V$ with $|\delta^{\text{out}}(v)| \ne 0$. Then for every edge $f = (u, v)$ there is an edge $e = (t, u)$ such that e and f are splittable.*

Bang-Jensen et al.'s exponential version of Theorem 3.1 repeatedly splits off unweighted pairs of edges and recurses on the new graph. We follow the same procedure but always split the same pair as many times/as much weight as possible at once. The following simple observation allows us to prove this runs in polynomial time.

Lemma B.4. *Let $D = (V, A)$ be an Eulerian digraph, $e = (t, u)$ and $f = (u, v)$ be splittable edges, $f' = (u, v')$ be an edge leaving u (possibly $f = f'$), and $X_{f'}$ be a tight set for some $x, y \ne u$ with $u \notin X_{f'}$, $v' \in X_{f'}$. Then $X_{f'}$ is still tight for x, y in D^{ef} after splitting off e and f.*

Proof. We have that $\lambda_{D^{ef}}(x,y) \geq \Lambda_D(x,y)$, since e and f are splittable and $x, y \neq u$. Splitting off cannot increase $\lambda(x,y)$, so $\Lambda_{D^{ef}}(x,y) = \Lambda_D(x,y)$. Splitting off does not affect $|\delta^{\text{in}}(X_{f'})|$ or $|\delta^{\text{out}}(X_{f'})|$ unless $t, v \in X_{f'}$, and by Lemma B.2 this cannot be the case since $X_{f'}$ is tight and e, f are splittable. Therefore $\Lambda_{D^{ef}}(x,y) = \min\{|\delta^{\text{in}}(X_{f'})|, |\delta^{\text{out}}(X_{f'})|\}$. ■

As a consequence, $O(n^2)$ splittings suffice to remove a node.

Lemma B.5. *Let u be a node in an Eulerian digraph $D = (V, A)$, and suppose we repeatedly choose t, v such that $(t,u), (u,v)$ are splittable and split them off to the maximum extent possible (i.e., maximum splittable weight). Then after $O(n^2)$ such splittings $|\delta^{\text{in}}(u)|$ and $|\delta^{\text{out}}(u)|$ will be reduced to 0.*

Proof. If $|\delta^{\text{out}}(u)| > 0$, then by Lemma B.3 there is a splittable pair $e = (t,u), f = (u,v)$. Splitting e, f as much as possible creates a tight set X_f, and this set remains tight after additional splittings centered at node u by Lemmas B.2 and B.4, so e, f will not become splittable again. After $O(n)$ splittings e', f for all possible e', w_f must be 0, and since there are $O(n)$ choices for f and $|\delta^{\text{in}}(u)| = |\delta^{\text{out}}(u)|$, $O(n^2)$ splittings suffice to remove all edges incident to u. ■

Proof of Theorem 3.1. Note that $r \notin U$. If $|U| \leq 1$ the theorem is trivial, so assume $|U| \geq 2$. For every $u \in U$, add an edge (u,r) of weight $|\delta^{\text{in}}(u)| - |\delta^{\text{out}}(u)|$, and let D' the resulting Eulerian graph (with $\lambda_{D'}(u,v) \geq \lambda_D(u,v)$).

Let $u = \text{argmin}_{v \in U} \Lambda(r,v)$, and $f = (u,v)$ be an edge leaving u with $w_f > 0$. By Lemma B.3 there exists an edge $e = (t,u)$ such that e and f are splittable. Let x be the maximum amount e and f are splittable, which can be found by binary search. Split off e and f to an extent x (subtract x from w_e, w_f, add x to $w_{(t,v)}$) and recurse on D'^{ef}.

Note that only $\Lambda_{D'^{ef}}(r,u)$ can decrease in the split, so in the recursive call we will choose the same u if $\Lambda_{D'^{ef}}(r,u) > 0$. By Lemma B.5 $O(n^2)$ iterations suffice to remove all edges incident to u. Future splittings after this centered on other nodes may create new edges but will never add an edge incident to a node of degree 0, so $O(n^3)$ splittings suffice.

We now undo the splitting to construct the arborescence packing on the original D. By induction we can find arborescences $F_1, \ldots, F_q$ in D^{ef} and weights $\gamma_1, \ldots, \gamma_q$ such that $\sum_{i=1}^q \gamma_i = K$, $\sum_{i:h \in F_i} \gamma_i \leq w_{h,D^{ef}}$ for all $h \in A(D^{ef})$, $\sum_{i:u' \in F_i} \gamma_i \geq \Lambda_D(r,u')$ for all $u' \neq u$, and $\sum_{i:u \in F_i} \gamma_i \geq \Lambda_{D^{ef}}(r,u) \geq \Lambda_D(r,u) - x$. First, we need to ensure that the F_i do not use the added arc $g = (t,v)$ above its weight in D, and second we need to update the arborescences to cover u to an additional x extent.

If $w_{g,D^{ef}} \geq \sum_{i:g \in F_i} \gamma_i > w_{g,D} = w_{g,D^{ef}} - x$ we need to decrease the use of g by $\sum_{i:g \in F_i} \gamma_i - w_{g,D}$, which is at most x. We can replace g with the pair e, f since $w_{h,D} - x = w_{h,D^{ef}} \geq \sum_{i:h \in F_i} \gamma_i$ for $h = e$ or f. Repeatedly choose F_i containing g until we have a set S with total weight at least x. Break the last F_i added to S into two identical arborescences with weights summing to γ_i, so that the set S has weight exactly x. This increases q by 1 (or $O(n^3)$, summing over all graphs in the induction).

For each $F_i \in S$, if $u \notin V(F_i)$, define $F_i' = F_i - g + e + f$. If $u \in V(F_i)$, let h be the last edge on the path P from r to u in F_i. If $g \notin P$, define $F_i' = F_i - g + f$, and if $a \in P$, define $F_i' = F_i - g - h + e + f$ (note F_i' is connected). Replace each F_i with F_i' in the arborescence packing. Over all $F_i \in S$, we remove g from trees with weight x and add e and f to trees with weight at most x. The updated $F_1', \ldots, F_{q+1}'$ now satisfy $\sum_{i:h \in F_i} \gamma_i \leq w_{h,D}$.

If $\Lambda_D(r,u) > \sum_{i:u \in F_i'} \gamma_i$ we need to increase the weight of some F_i' containing u. By assumption $\sum_{i:u \in F_i} \gamma_i \geq \Lambda_{D^{ef}}(r,u) \geq \Lambda_D(r,u) - x$. We chose u such that $\Lambda_D(r,u) \leq \lambda_D(r,v)$ for all v, so $\Lambda_{D^{ef}}(r,u) \leq \lambda_{D^{ef}}(r,v) - x$. Therefore for every $v \neq u$ there are F_i containing v but not u with total weight at least x (or $x - (\sum_{i:u \in F_i'} \gamma_i - \Lambda_{D^{ef}}(r,u))$ if u is in more F_i than required).

Let S_2 be a set of F_i' with weight at least x containing t but not u. Add the edge e to $F_i' \in S_2$. In the process we may exhaust the budget $w_e - \sum_{i:u \in F_i'} \gamma_i$ for e due to adding e to some $F_i \in S$ in the previous

step. This can happen for two reasons. In the first case, there was some $F_i \in S$ with $u \notin V(F_i)$, and we defined $F_i' = F_i - g + e + f$. But that change also increased the weight of F_i' containing u beyond $\Lambda_{Def}(r,u)$ and is not a problem.

The second case is that $u \in F_i$, and we set $F_i' = F_i - g - h + e + f$, which uses budget for e even though u is already in F_i'. However, at the same time we lost y budget for e, we freed up y budget for some $h = (s,u)$, and we can find a set of F_i with weight y containing s but not u and add the edge h. This step may require breaking some F_i into two trees with total weight γ_i that are identical except that one contains u and the other does not, which may increase q by $O(n)$, but it remains polynomially bounded ($O(n^4)$ added over the entire induction). ■

C Proof of Corollary 3.3

Proof of part (i). We utilize part (ii) of Theorem 3.2. We may assume that when $\lambda = L = 0$, the tree T_λ returned is the trivial tree consisting only of $\{r\}$, and when λ is very large, say $H = nc_{\max}$, then T_λ spans all nodes. So if $B = 0$ or n, then we are done, so assume otherwise. Let $\mathcal{C}_B^*$ be an optimal collection of rooted paths, so $O^* = \sum_{P \in \mathcal{C}_B^*} c(P)$. Let $n^* = |\bigcup_{P \in \mathcal{C}_B^*} V(P)| \geq B$. We preform binary search in $[L, H]$ to find a value λ such that $|V(T_\lambda)| = B$. We maintain the invariant that we have trees T_1, T_2 for the endpoints $\lambda_1 < \lambda_2$ of our search interval respectively such that $|V(T_1)| < B < |V(T_2)|$ and $c(T_i) + \lambda_i |V \setminus V(T_i)| \leq O^* + \lambda_i(n - n^*)$ for $i = 1, 2$. Let $\lambda = (\lambda_1 + \lambda_2)/2$, and let $T = T_\lambda$ be the tree returned by Theorem 3.2 (ii). If $|V(T)| = B$, then we are done and we return the rooted tree T. Otherwise, we update $\lambda_2 \leftarrow \lambda$ if $|V(T)| > B$, and update $\lambda_1 \leftarrow \lambda$ otherwise.

We terminate the binary search when $\lambda_2 - \lambda_1$ is suitably small. To specify this precisely, consider the parametric LP (PC-LP) where $\pi_v = \lambda$ for all $v \in V$ and λ is a parameter. We say that λ is a *breakpoint* if there are two optimal solutions (x^1, z^1), (x^2, z^2) to (PC-LP) with $\sum_v z_v^1 \neq \sum_v z_v^2$. (This is equivalent to saying that the slope of the optimal-value function is discontinuous at λ.) We may assume that (x^1, z^1), (x^2, z^2) are vertex solutions and so their non-zero values are multiplies of $\frac{1}{M}$ for some M (that can be estimated) with $\log M = \text{poly}(\text{input size})$. But then $\sum_e c_e x_e$ and $\sum_v z_v$ are also multiples of $\frac{1}{M}$ for both solutions (since the c_es are integers), and hence the breakpoint λ is a multiple of $\frac{1}{M'}$, for some $M' \leq nM$. We terminate the binary search when $\lambda_2 - \lambda_1 < \frac{1}{2n^2 M}$. Observe that the binary search takes polynomial time.

So if we do not find λ such that $|V(T_\lambda)| = B$, at termination, we have that $c(T_i) + \lambda_i(n - |V(T_i)|) \leq O^* + \lambda_i(n - n^*)$ for $i = 1, 2$. There must be exactly one breakpoint $\lambda \in [\lambda_1, \lambda_2]$. There must be at least one breakpoint since $T_1 \neq T_2$, which can only happen if the optimal solutions to (PC-LP) differ for λ_1 and λ_2, and there cannot be more than one breakpoint since any two breakpoints must be separated by at least $\frac{1}{nM}$ as reasoned above.

We claim that we have $c(T_i) + \lambda(n - |V(T_1)|) \leq O^* + \lambda(n - n^*)$ for $i = 1, 2$. If we show this, then taking a, b so that $a|V(T_1)| + b|V(T_2)| = B$, $a + b = 1$, and taking the (a,b)-weighted combination of the two inequalities, we obtain that $ac(T_1) + bc(T_2) + \lambda(n - B) \leq O^* + \lambda(n - n^*)$, and so we are done. (Note that we do *not* actually need to find the breakpoint λ.)

To prove the claim observe that

$$\begin{aligned} c(T_1) + \lambda(n - |V(T_1)|) &\leq c(T_1) + \lambda_1(n - |V(T_1)|) + \frac{1}{2nM} \\ &\leq O^* + \lambda_1(n - n^*) + \frac{1}{2nM} \leq O^* + \lambda(n - n^*) + \frac{1}{2nM}. \end{aligned}$$

So $\big[c(T_1) + \lambda(n - |V(T_1)|)\big] - \big[O^* + \lambda(n - n^*)\big] \leq \frac{1}{2nM}$, but the LHS is a multiple of $\frac{1}{M'}$, so the LHS must be nonpositive. A similar argument shows that $c(T_2) + \lambda(n - |V(T_2)|) \leq O^* + \lambda(n - n^*)$. ■

Proof of part (ii). We mimic the proof of part (i), and only discuss the changes. Assume that $0 < C <$(cost of MST of $\{v : w_v > 0\}$) to avoid trivialities. Let $\mathcal{W}_C^*$ be an optimal collection of rooted paths, so $n^* = w(\bigcup_{P\in\mathcal{W}_C^*} V(P))$. Let $O^* = \sum_{P\in\mathcal{W}_C^*} c(P) \le C$. Let K be such that all w_vs are multiples of $\frac{1}{K}$; note that $\log K = \text{poly}(\text{input size})$. Let $W = \sum_v w_v$. For a given parameter λ, we now consider (PC-LP) with penalties λw_v for all v. We perform binary search in $[L = 0, H = KWc_{\max}]$; we may again assume that T_L is the trivial tree, and T_H spans all nodes with positive weight. Given the interval $[\lambda_1, \lambda_2]$, we maintain that the trees T_1, T_2 for λ_1, λ_2 satisfy $c(T_1) < C < c(T_2)$ and $c(T_i) + \lambda_i w(V \setminus V(T_i)) \le O^* + \lambda_i(W - n^*)$ for $i = 1, 2$. As before, we find tree T_λ for $\lambda = (\lambda_1 + \lambda_2)/2$. If $c(T_\lambda) = C$ then $w(V(T_\lambda)) \ge n^*$ and we are done and return T_λ. Otherwise, we update $\lambda_2 \leftarrow \lambda$ if $c(T) > C$, and $\lambda_1 \leftarrow \lambda$. We terminate when $\lambda_2 - \lambda_1 \le \frac{1}{2W^2K^2M}$.

Similar to before, one can argue that every breakpoint of the parametric LP with penalties $\{\lambda w_v\}$ must be a multiple of $\frac{1}{M'}$ for some $M' \le MKW$. So at termination (without returning a tree), there is a breakpoint $\lambda \in [\lambda_1, \lambda_2]$. We have that for $i = 1, 2$,

$$\Big[c(T_i) + \lambda\big(W - w(V(T_i))\big)\Big] - \Big[O^* + \lambda(W - n^*)\Big] \le \frac{1}{2WK^2M}$$

but is also a multiple of $\frac{1}{KM'}$, so the above quantity must be nonpositive for $i = 1, 2$. Let a, b be such that $a + b = 1$ and $ac(T_1) + bc(T_2) = C$. Then, we have

$$a\Big[c(T_1) + \lambda\Big(W - w\big(V(T_1)\big)\Big)\Big] + b\Big[c(T_2) + \lambda\Big(W - w\big(V(T_2)\big)\Big)\Big] \le O^* + \lambda(W - n^*).$$

So $aT_1 + bT_2$ yields the desired bipoint tree. ■

D Weighted sum of latencies: a combinatorial 7.183-approximation for k-MLP

We may assume via scaling that all weights are integers, and $w_r = 1$. Let $W = \sum_{u\in V} w_u$. As before, let $\mathsf{TS} := \{\mathsf{T}_{-1} = 0, \mathsf{T}_0, \ldots, \mathsf{T}_D\}$, where $\mathsf{T}_j = \lceil (1+\epsilon)^j \rceil$ for all $j \ge 0$, and $D = O(\log \mathsf{T}) = \text{poly}(\text{input size})$ is the smallest integer such that $\mathsf{T}_D \ge \mathsf{T}$. We think of applying the concatenation-graph argument by considering time points in the polynomially-bounded set TS. We obtain a suitable k-tuple of tours for each of these time points, and concatenate the tours corresponding to the shortest path in the corresponding concatenation graph.

As in Algorithm 3, we initialize $C \leftarrow \{(1, 0)\}$ and $\mathcal{Q} \leftarrow \emptyset$. For each time $t = 0, 1, \ldots$, let wt_t denote the maximum node weight of a collection of k rooted paths, each of length at most t. Note that $\mathsf{wt}_0 = 1$. By part (ii) of Corollary 3.3, given t, we can efficiently compute a rooted tree Q_t or rooted bipoint tree (a_t, Q_t^1, b_t, Q_t^2) of cost kt and node weight at least wt_t. We compute these objects for all $t \in \mathsf{TS}$. Say that t is single or bipoint depending on whether we compute a tree or a bipoint tree for t respectively. Observe that given a rooted tree Q, one can obtain a collection of k cycles, each of length at most $\frac{2c(Q)}{k} + 2\max_{v\in V(Q)} c_{rv}$, that together cover $V(Q)$.

For each $t \in \mathsf{TS}$, if t is single, we include the point $\big(w(V(Q_t)), \frac{2c(Q_t)}{k} + 2t\big)$ if t is single and add Q_t to $\mathcal{Q}$; if t is bipoint, we add the points $\big(w(V(Q_t^1)), \frac{2c(Q_t^1)}{k} + 2t\big)$, $\big(w(V(Q_t^2)), \frac{2c(Q_t^2)}{k} + 2t\big)$ to C and add the trees Q_t^1, Q_t^2 to $\mathcal{Q}$. Let $f : [1, W] \mapsto \mathbb{R}_+$ be the lower-envelope curve of C. Note that $f(1) = 0$ and $f(W) \le 4\mathsf{T}_D$. By Theorem 2.1, the shortest path P_C in the concatenation graph $CG\big(f(0), f(1), \ldots, f(W)\big)$ only uses nodes corresponding to corner points of f, all of which must be in C. So P_C can be computed by finding the shortest path in the polynomial-size (and polytime-computable) subgraph of the concatenation graph consisting of edges incident to nodes corresponding to corner points of f.

We now argue that we can obtain a solution of cost at most the length of P_C in the concatenation graph. Suppose that (o, ℓ) is an edge of the shortest path. Since $\big(\ell, f(\ell)\big)$ is a corner point, we have a tree $Q_\ell \in \mathcal{Q}$

and time $t_\ell \in \mathsf{TS}$ such that: $w(V(Q_\ell)) = \ell$, $f(\ell) = \frac{2c(Q_\ell)}{k} + 2t_\ell$, and $\max_{v \in V(Q_\ell)} c_{rv} \le t_\ell$. Thus, as noted earlier, we can obtain from Q_ℓ a collection of k cycles, each of length at most $f(\ell)$ that together cover nodes of total weight ℓ. One can then mimic the inductive argument in Theorem 6.1 to prove the claimed result.

By Theorem 2.1, the length of P_C is at most $\frac{\mu^*}{2} \sum_{\ell=1}^{W} f(\ell)$. Define $b^*_\ell = \min\{t : \mathsf{wt}_t \ge \ell\}$, similar to the definition of $\mathsf{BNS}(k, \ell)$. So $\sum_{\ell=1}^{W} b^*_\ell$ is a lower bound on the optimal value. For an integer $x \ge 0$, recall that $[x]$ denotes $\{1, \ldots, x\}$; let $[\![x]\!]$ denote $\{0\} \cup [x]$. Define $\mathsf{T}_{-1} := 0$. Dovetailing the proof of Lemma 6.2, we have

$$\frac{\sum_{\ell=1}^{W} f(\ell)}{4} = \int_{t=0}^{f} (W) dt \big|\{\ell \in [W] : f(\ell) \ge 4t\}\big| \le \sum_{j=0}^{D} (\mathsf{T}_j - \mathsf{T}_{j-1}) \big|\{\ell \in [W] : f(\ell) > 4\mathsf{T}_{j-1}\}\big|. \quad (20)$$

Consider some $t \in \mathsf{TS}$. Recalling how the point-set C is produced, we observe that:

(a) if t is single, then $(w, 4t) \in C$ for some $w \ge \mathsf{wt}_t$;

(b) if t is bipoint, then $(w, 4t)$ lies in the convex hull of C for some $w \ge \mathsf{wt}_t$.

In both cases, this implies that $f(\mathsf{wt}_t) \le 4t$. So $\big|\{\ell \in [W] : f(\ell) > 4\mathsf{T}_{j-1}\}\big| \le W - \mathsf{wt}_{\mathsf{T}_{j-1}}$ for $j \ge 1$; this also holds when $j = 0$ since $f(1) = 0$, $\mathsf{wt}_0 = 1$. Substituting this in (20), gives that $\frac{\sum_{\ell=1}^{W} f(\ell)}{4}$ is at most

$$\sum_{j=0}^{D} (\mathsf{T}_j - \mathsf{T}_{j-1})\big(W - \mathsf{wt}_{\mathsf{T}_{j-1}}\big) = \sum_{j=0}^{D} (\mathsf{T}_j - \mathsf{T}_{j-1}) \sum_{\ell=\mathsf{wt}_{\mathsf{T}_{j-1}}+1}^{W} 1 = \sum_{\ell=2}^{W} \sum_{j \in [\![D]\!] : \mathsf{wt}_{\mathsf{T}_{j-1}}+1 \le \ell} (\mathsf{T}_j - \mathsf{T}_{j-1}) = \sum_{\ell=2}^{W} \mathsf{T}_{B_\ell}$$

where $B_\ell = \min\{j \in [\![D]\!] : \mathsf{wt}_{\mathsf{T}_j} \ge \ell\}$. Clearly $\mathsf{T}_{B_\ell} \le (1+\epsilon) b^*_\ell$, so $\frac{\sum_{\ell=1}^{W} f(\ell)}{4} \le (1+\epsilon) \sum_{\ell=1}^{W} b^*_\ell$.

E Node service times: a combinatorial 7.183-approximation for k-MLP

Recall that we define the mixed length of a path or tree Q to be $c(Q) + d(V(Q))$, and c' is the directed metric $c'_{u,v} = c_{uv} + d_v$ for all u, v. So the c'-cost of an out-tree rooted at r is exactly its mixed length (since $d_r = 0$). The changes to Algorithm 3 are as follows.

- First, we sort nodes in increasing order of $c_{rv} + d_v$; let $u_1 = r, u_2, \ldots, u_n$ be the nodes in this sorted order, and let $H_j = (U_j, E_j)$ be the subgraph induced by $U_j := \{u_1, \ldots, u_j\}$.
- In step S2, we use part (i) of Corollary 3.3 with the graph H_j and the mixed-length objective. If we get a rooted out-tree $Q_{j\ell}$, we add the point $\big(|V(Q_{j\ell})|, \frac{2c'(Q_{j\ell})}{k} + 2c'_{r,u_j}\big)$ to C. If we get a rooted bipoint out-tree, we add the points $\big(|V(Q^1_{j\ell})|, 2\frac{c'(Q^1_{j\ell})}{k} + 2c'_{r,u_j}\big)$ and $\big(|V(Q^2_{j\ell})|, 2\frac{c'(Q^2_{j\ell})}{k} + 2c'_{r,u_j}\big)$ to C. As before, we add the rooted tree or the constituents of the bipoint tree to $\mathcal{Q}$.
- In step S5, we obtain a collection of k tours $Z_{1,\ell}, \ldots, Z_{k,\ell}$ from the rooted tree Q^*_ℓ by applying Lemma 7.2 on Q^*_ℓ with $S = V(Q^*_\ell)$ to obtain k cycles, and traverse each cycle in a random direction. The resulting tours satisfy

$$c(Z_{i,\ell}) + 2d(V(Z_{i,\ell})) \le 2 \cdot \frac{c(Q^*_\ell) + d\big(V(Q^*_\ell)\big)}{k} + 2\big(c_{ru_{j^*_\ell}} + d_{u_{j^*_\ell}}\big) \qquad \text{for all } i = 1, \ldots, k. \quad (21)$$

The remaining steps of Algorithm 3 (i.e., S1, S3, S4, S6) are unchanged.

The analysis follows the one in Section 6.2. The bottleneck-(k, ℓ)-stroll problem is now defined with respect to the mixed-length objective; so $\mathsf{BNS}(k, \ell)$ is the smallest L such that there are k rooted paths $P_1, \ldots, P_k$, each of mixed length at most L that together cover at least ℓ nodes. As before, $\mathsf{BNSLB} := \sum_{\ell=1}^{n} \mathsf{BNS}(k, \ell)$ is a lower bound on the optimum.

Lemma 6.5 holds as is. The argument for part (ii) is unchanged. For part (i), suppose that v_j is the node furthest from r that is covered by some optimal (k, ℓ)-bottleneck-stroll solution, so $c_{ru_j} + d_{u_j} \leq \mathsf{BNS}(k, \ell)$. Then, in iteration (j, ℓ) of step S2, we add one or two points to C such that the point $\left(\ell, \frac{2z}{k} + 2(c_{ru_j} + d_{u_j})\right)$, for some $z \leq k \cdot \mathsf{BNS}(k, \ell)$, lies in the convex hull of the points added. Therefore, $s_\ell = f(\ell) \leq 4 \cdot \mathsf{BNS}(k, \ell)$ since f is the lower-envelope curve of C. Finally, the proof that the cost of the solution returned is at most the length of the shortest path in $CG(s_1, \ldots, s_n)$ is essntially identical to the proof in the LP-rounding 7.183-approximation algorithm for k-MLP in Section 7.3.