Minimizing Latency in Online Ride and Delivery Services*

Abhimanyu Das
Google Research
abhidas@google.com

Sreenivas Gollapudi
Google Research
sgollapu@google.com

Anthony Kim†
Stanford University
tonyekim@stanford.edu

Debmalya Panigrahi‡
Duke University
debmalya@cs.duke.edu

Chaitanya Swamy§
University of Waterloo
cswamy@uwaterloo.ca

ABSTRACT

Motivated by the popularity of online ride and delivery services, we study natural variants of classical multi-vehicle minimum latency problems where the objective is to route a set of vehicles located at depots to serve requests located on a metric space so as to minimize the total latency. In this paper, we consider point-to-point requests that come with source-destination pairs and release-time constraints that restrict when each request can be served. The point-to-point requests and release-time constraints model taxi rides and deliveries. For all the variants considered, we show constant-factor approximation algorithms based on a linear programming framework. To the best of our knowledge, these are the first set of results for the aforementioned variants of the minimum latency problems. Furthermore, we provide an empirical study of heuristics based on our theoretical algorithms on a real data set of taxi rides.

CCS CONCEPTS

• **Theory of computation → Routing and network design problems**; *Discrete optimization*;

KEYWORDS

Vehicle Routing; Minimum Latency Problem; Online Ride Services

ACM Reference Format:
Abhimanyu Das, Sreenivas Gollapudi, Anthony Kim, Debmalya Panigrahi, and Chaitanya Swamy. 2018. Minimizing Latency in Online Ride and Delivery Services. In *WWW 2018: The 2018 Web Conference, April 23–27, 2018, Lyon, France.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3178876.3186104

1 INTRODUCTION

In recent years, ride-sharing platforms such as Lyft, Ola and Uber, and online delivery services such as DoorDash and GrubHub have become increasingly popular and have expanded their operations to many cities and countries. A central problem common to these online services is the vehicle routing problem where a fleet of vehicles are routed to serve ride and delivery requests over a geographical area. Indeed, this problem is also at the core of traditional city taxi services, such as Yellow Cab, where taxis are routed to serve ride requests received over the phone or the Internet. In all these settings, a request comprises a pair of source and destination locations, such as the rider's starting and ending coordinates for taxi services, and the restaurant and customer addresses for food delivery services. Furthermore, these "point-to-point" requests typically have release-time constraints, i.e., the customer specifies a desired service time before which the request cannot be serviced. The vehicle routing algorithm desires to minimize the average latency of customers, i.e., the difference between their requested service time and the actual service time.

In this paper, we formally define a multi-vehicle routing problem, obtain an algorithm based on a linear programming framework with a formal guarantee on its performance, and demonstrate that heuristics based on this formal algorithm improve on benchmark greedy solutions on real data sets of city taxi rides.

Most current systems, such as the online[1] and traditional services mentioned above, employ various heuristics for a batch of requests to solve this kind of vehicle routing problems in practice. In contrast, there is a rich history in the algorithmic literature on the so-called *vehicle routing problems* (or VRP), which covers a wide range of routing problems for one or more vehicles under a variety of constraints. For these problems, the formal literature comprises a wide array of sophisticated techniques, often based on linear programing formulations, that lead to approximation algorithms with provable guarantees.

While we are not aware of any previous results on our exact problem formulation with point-to-point requests and release-time constraints, the related literature raises the natural question: can these sophisticated algorithmic techniques be brought to bear on this important practical problem of minimizing latency for point-to-point requests with release-time constraints? And if so, do these algorithmic ideas also lead to better heuristics in practice? We answer both these questions affirmatively by designing a constant approximation algorithm for this problem, which also leads to a heuristic that outperforms a natural greedy strategy.

*The full version of the paper is available at https://arxiv.org/abs/1802.02744.
†Part of this work was done when the author was an intern at Google, Inc.
‡Supported in part by NSF grants CCF-1535972, CCF-1527084, and a CAREER Award.
§Supported in part by NSERC grant 327620-09 and an NSERC Discovery Accelerator Supplement award

[1]Note that we use "online" to refer to the requests being generated by web-based services, and not to refer to requests arriving "online" in an algorithmic sense.

1.1 Our Contributions

We formally define the minimum latency problems in Section 2. In Section 3, we present the linear programming framework due to Post and Swamy [23] that will be central in our algorithms and analyses. Our theoretical contributions are to obtain **constant approximation algorithms** for the following problems:

(1) (Section 4.1) For the minimum latency problem (MLP) and single-depot, multi-vehicle minimum latency problem (k-MLP) with point-to-point requests.
(2) (Section 4.2) For the multi-depot, multi-vehicle minimum latency problem (k-MLP) with point-to-point requests.
(3) (Section 5) For the above problems with release-time constraints for both point and point-to-point requests.

Additionally, we perform a large-scale empirical analysis for the minimum latency problem by comparing our algorithms with a natural greedy baseline using two real-world taxi data sets. We show that our algorithms outperform the baseline on a set of different metrics including latency, tour length, and utilization (active time) of the cabs in the system.

See Table 1 for a summary of our theoretical results with the precise constant approximation factors. To the best of our knowledge, *these results are the first polynomial-time approximation guarantees for the respective minimum latency problems.* For multi-depot problems, we obtain approximation guarantees with somewhat large constants via a constant-factor reduction (of ratio 3). We believe better approximation ratios are possible with use of more complicated constructs, but we do not explore these in this paper to keep our algorithms viable in practice.

"Client-side" vs "Platform-side" objectives: Note the average latency objective that we study in this paper is a "client-side" objective whereas the total distance traveled by the vehicles, commonly studied in problems such as the well-known *traveling salesman problem* (or TSP), is a "platform-side" objective. We can interpret the average latency as the average waiting time from the clients' perspective and the total distance as the platform's operation cost for fuel, etc. Our routing problem with point-to-point requests can be thought of as the "client-side" counterpart of what is known as the Dial-a-Ride problem with unit-capacity vehicles, where vehicles serve requests with point-to-point requests as in our problem, but seek to minimize the "platform-side" objective of total travel distance. The latency minimization problem with point requests subject to release-time constraints has been proposed as an open problem in [32] and there exist polynomial-time approximation schemes for special cases such as a constant number of vehicles, or if the metric space has a special structure such as the Euclidean plane or weighted trees [30]. Our result is the first one for a general metric space and an arbitrary number of vehicles, and generalizes further to point-to-point requests.

1.2 Related Work

Our problem is an example of a vehicle routing problem, which is a generic term used to describe a wide range of routing problems over metric spaces. Of particular relevance to our context is the *minimum latency problem* (or MLP), also known as the traveling repairman problem or the delivery man problem, which has applications in diskhead scheduling and searching information in a network such as the world wide web [7, 17]. This problem is a special case of our problem, where requests are at point locations (instead of point-to-point requests) and there are no release-time constraints. MLP and k-MLP (respectively, one or k vehicles) have been long studied in both the Operations Research and Computer Science communities. MLP was shown to be NP-complete and then MAXSNP-hard for general metrics [8, 22, 26]. In fact, it is NP-hard even when the metric is a weighted tree with $\{0, 1\}$ weights [29] and is thought to be harder than the well-known traveling salesman problem.[2]

There have been many works focused on exactly solving MLP/k-MLP and related problems, albeit not in polynomial time. A number of mixed integer formulations have been proposed and exact methods such as cutting-plane algorithms and branch-cut-and-price algorithms (e.g., [1, 18, 19]) and various meta-heuristics (e.g., [20, 21, 28]) have been proposed. More recently, several mixed integer formulations for k-MLP have been proposed and experimented with on routing and scheduling instances with the number of nodes ranging up to 80 [3].

The first constant-factor approximation algorithm for MLP was obtained in [8] and the approximation factor was subsequently improved to 3.592 in a series of work [5, 9, 15]. Similarly, we have constant-factor approximation algorithms for the multi-vehicle version, k-MLP, for both the multi-depot and single-depot variants due to a series of work [9, 11, 14, 23]: the current best-known approximation factors are 8.497 and 7.183 respectively. For several special cases, stronger guarantees are known. A quasi-polynomial time approximation scheme was known for weighted trees and Euclidean metrics in any finite dimensions [6], and, more recently, a polynomial time approximation scheme was shown for weighted trees and the Euclidean plane for MLP and single-depot k-MLP for any constant k, i.e., a constant number of vehicles [30].

MLP/k-MLP are also closely related to other vehicle routing problems such as the traveling salesman problem, orienteering (cf. [10]) and the Dial-a-Ride problem (cf. [13]). They are also related to many sequencing problems with the minimum total (weighted) completion time objective in the scheduling literature (cf. [16, 30]). There is a large body of work on vehicle routing problems and scheduling problems beyond the scope of this paper. For further details, we refer to the above work and references therein.

Finally, we mention several work among many on other aspects of the ride and delivery services from the Data Mining and Artificial Intelligence communities. Different taxi dispatching strategies and route-recommendation systems have been studied in order to minimize passengers' waiting times (cf. [2, 34] in different settings from ours), to maximize drivers' profits (cf. [25]), and to guarantee fairness within a group of competing drivers (cf. [24]). To address inefficiencies in taxi systems, several graph-based models and algorithms have been designed to minimize the total number of required taxis and to reduce the total idle time of taxi drivers (cf. [33, 35]). Dynamic variants where ride requests arrive on demand have also been studied (cf. [27]).

[2]The traveling salesman problem is MAXSNP-hard for general metrics but is solvable in polynomial time in the case of weighted tree metrics (via an Eulerian tour).

	P Reqs.	P2P Reqs.	P Reqs. w/ RTs	P2P Reqs. w/ RTs
MLP	3.592 ([9])	3.592	7.183	7.183
Single-Depot k-MLP	7.183 ([23])	8.978	7.183	8.978
Multi-Depot k-MLP	8.497 ([23])	25.488	13.728	41.184

Table 1: The state-of-the-art approximation guarantees for various minimum latency problems (MLP/k-MLP) with point (P) requests and point-to-point (P2P) requests and with or without release times (RTs). Except those in the first column, the constant-factor approximation ratios are new and due to this paper.

2 PROBLEM FORMULATION

We define the multi-vehicle minimum latency problem (k-MLP) with point-to-point requests as follows. Let $G = (V, E)$ be a weighted complete undirected graph with a distance function on edges, $c : E \to \mathbb{R}^+$, that forms a metric space. There are n point-to-point requests where each request R_i is given by a pair (s_i, d_i) of source and destination nodes and to be satisfied without interruption, that is, a vehicle serves it by first going to the source node and then to the destination node directly. There are k vehicles located at respective designated depot nodes, equivalently, root nodes, $r_1, \dots, r_k$. The objective is to find k paths $P_1, \dots, P_k$ starting from respective depots that serve all the requests and minimize the *total latency*, that is, the sum of latencies of requests, where the latency of a request is equal to the distance from the depot to the destination of the request on the path of the vehicle that serves it.[3]

After appropriate scaling and rounding, we may assume c_e are integers and $c_{r_i s_j} + c_{s_j d_j} \geq 1$ for every root node r_i and request R_j. For ease of exposition, we assume the distances c_e are given in a unit of time and interpret the latency of a request to be its completion time, following the job scheduling literature.

The problem thus defined is the most general version to be studied in this paper and we refer to it as *multi-depot k-MLP with point-to-point requests.* We refer to the case when there is a single depot r_0 for all vehicles (i.e., $r_0 = r_1 = \cdots = r_k$) as *single-depot k-MLP with point-to-point requests*, and the case with exactly one vehicle (i.e., $k = 1$) as *MLP with point-to-point requests.* When the requests' source and destination nodes are identical, we have the classical k-MLP and we refer to them as *(multi-depot/single-depot) k-MLP with point requests* (e.g., [23]) to distinguish them from the more general problems with point-to-point requests.

MLP/k-MLP *with release times* is MLP/k-MLP with additional release-time constraints. Each request R_i has release time T_i such that it cannot be visited before time T_i. Both point and point-to-point requests can be considered with release times. When a vehicle is at the source node of a request, it may wait there until when the request becomes available at its release time. When clear from the context, we refer to these problems with shorter names.

Notations. We refer to the designated nodes using the r_i, s_i and d_i, and to arbitrary nodes of any kind using generic indexing variables such as u and v. We use $\mathcal{I}$ to denote the input size. A path may start from and end at different nodes and a tour must start from and end at the same node. We represent a path/tour by the sequence of nodes on the path/tour or simply by the sequence of requests in the order served if it is a vehicle's route. For example, if the order of the requests is $R_1 \cdots R_n$, the corresponding path is $P = r_0 s_1 d_1 \cdots s_n d_n$ (assuming the root r_0). For a path P, let $\text{Lat}(P, i)$ be the latency of the i-th request on the path, i.e., the distance along the path from the root of the path to the destination of the request, and $\text{Lat}(P) = \sum_{i=1}^{n} \text{Lat}(P, i)$ be the total latency of the path. The total latency objective is the sum of the latencies of the paths of the k vehicles. Note that if there is one vehicle located at the root r_0 that serves the all the requests in the order $R_1 \cdots R_n$, then

$$\text{Lat}(P, i) = \begin{cases} \text{Lat}(P, i-1) + c_{d_{i-1} s_i} + c_{s_i d_i}, & i > 1 \\ 0, & i = 0 \end{cases},$$

where $d_0 = r_0$.

Given a set of edges Q and a distance function c, let $c(Q)$ be the sum of the lengths given by c of the edges in Q; if P is a path, then $c(P)$ is the length of the path. We frequently perform concatenation and shortcutting operations on paths and tours. If two paths/tours meet at the same node, that is, one ending at and the other starting from the same node, we may concatenate them back to back to create a longer path/tour. We may shortcut a path/tour to avoid visiting a node twice by skipping it; this leads to a path/tour of at most the original length if the distances satisfy the triangle inequality, which holds in metric spaces.

3 LP FRAMEWORK

We describe the linear programming framework due to Post and Swamy [23] for single-depot k-MLP and MLP with point requests. Some of our algorithms utilize a directed metric, so we describe their LP in this general setting. Linear programs for multi-depot k-MLP are also given in [23] and are different, but we primarily focus on the linear program for the single-depot case in this paper. Let r denote the single depot for the point-request version.

Given a problem instance with point requests, let digraph $D = (V, A)$ with arc-costs $\{c_{u,v}\}_{u,v \in V}$ represent the underlying directed metric. (If we have an undirected metric, we simply bidirect the edges to obtain D, setting $c_{u,v} = c_{v,u} = c_{uv}$.) We use a to index the arcs in A, v to index nodes in $V \setminus \{r\}$, i to index the k vehicles, and t to index time units in $[\mathsf{T}]$. We use variables $x^i_{v,t}$ to denote if node v is visited at time t by the route originating at the root. Directing the vehicles' routes away from the root in a solution, $z^i_{a,t}$ indicates if arc a lies on the portion of vehicle i's route up to time t. Let T be an easily certifiable upper bound on the maximum latency of a request. Consider the following LP. We will be able to ensure that either: (a) T is polynomially bounded by scaling and rounding the metric (e.g., [6]) while losing a $(1 + \epsilon)$-factor, in which case this LP can be solved efficiently; or (b) $\log \mathsf{T}$ is polynomially bounded, and use ideas from [23] to obtain a $(1 + \epsilon)$-approximate solution to this LP.

Constraints (1) ensure that every non-root node is visited at some time, and constraints (2) ensure that each node cannot be visited

[3]Note the total latency and average latency differ by the factor of n and optimizing with respect to both objectives are equivalent.

before the distance from the root is covered. Constraints (3)–(5) are for the vehicles' routes: (3) ensures that the portion of a vehicle's route up to time t must visit every node visited by that vehicle by time t, (4) ensures that this route indeed has length at most t, and finally (5) seeks to encode that the route forms a path. (Note that constraints (5) are clearly valid, and one could also include the constraints $\sum_{a\in\delta^{\text{out}}(r)} z^i_{a,t} \le 1$ for all i, t.)

$$
\begin{aligned}
\min \quad & \sum_{v,t,i} t x^i_{v,t} && && \text{(LP)}\\
\text{s.t.} \quad & \sum_{t,i} x^i_{v,t} \ge 1 && \forall v && (1)\\
& x^i_{v,t} = 0 \text{ if } c_{r,v} > t && \forall v,t,i && (2)\\
& \sum_{a\in\delta^{\text{in}}(S)} z^i_{a,t} \ge \sum_{t'\le t} x^i_{v,t'} && \forall S \subseteq V\setminus\{r\}, v\in S, \forall t,i && (3)\\
& \sum_a c_a z^i_{a,t} \le t && \forall t,i && (4)\\
& \sum_{a\in\delta^{\text{in}}(v)} z^i_{a,t} \ge \sum_{a\in\delta^{out}(v)} z^i_{a,t} && \forall v,t,i && (5)\\
& x,z \ge 0. && && (6)
\end{aligned}
$$

To round a fractional solution to a set of routes for vehicles, we use a polynomial-time arborescence packing result for weighted digraphs and the concatenation graph. The following result does not require the edge costs are symmetric or form a metric, but holds for arbitrary nonnegative edge costs.

Theorem 1 (Theorem 3.1 in [23]). *Let $D = (U+r, A)$ be a digraph with nonnegative integer edge weights $\{w_e\}$, where $r \notin U$ is a root node and $|\delta^{\text{in}}(u)| \ge |\delta^{\text{out}}(u)|$ 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 integers $\gamma_1, \ldots, \gamma_q$ 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 $\sum_{i:u\in F_i} \gamma_i = \min\{K, \lambda_D(r,u)\}$ for all $u \in U$.*

The *concatenation graph* was introduced by Goemans and Kleinberg [15] and then extended by Archer and Blasiak [4] as a convenient mean of representing the concatenation process of constructing vehicles' routes from shorter paths. The concatenation graph corresponding to a sequence $w_1 = 0, \ldots, w_n$ of nonnegative numbers (that starts with a 0), denoted $CG(w_1, \ldots, w_n)$, is a directed graph with n nodes and an arc (i, j) of length $\left(n - \frac{i+j}{2}\right) w_j$ for all $i < j$. In our applications, a path through $CG(w_1, \ldots, w_n)$ will correspond to the selection of certain partial solutions of smaller routes and the subsequent concatenation of these partial solutions to obtain a final solution of the vehicles' routes. The length of the path will upper bound the total latency of the final solution.

We say that w_ℓ is an *extreme point* of the sequence $(w_1, \ldots, w_n)$ if (ℓ, w_ℓ) is an extreme-point of the convex hull of $\{(j, w_j) : j = 1, \ldots, n\}$. Given a point-set $C \subseteq \mathbb{R}^2_+$, define its *lower-envelope curve* $f : [\min_{(x,y)\in C} x, \max_{(x,y)\in C} x] \mapsto \mathbb{R}_+$ by $f(x) = \min\{y : (x,y) \in \text{conv}(C)\}$ where $\text{conv}(C)$ denotes the convex hull of C. We say that (ℓ, w_ℓ) is a *corner point* of the lower-envelope curve of $\{(j, w_j) : j = 1, \ldots, n\}$ if w_ℓ is an extreme point of $(w_1, \ldots, w_n)$.

We have the following results on the concatenation graph:

Theorem 2 ([4, 15]). *The shortest $1 \rightsquigarrow n$ path in $CG(w_1, \ldots, w_n)$ has length at most $\frac{\mu^*}{2} \sum_{\ell=1}^n w_\ell$ 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 $(w_1, \ldots, w_n)$.*

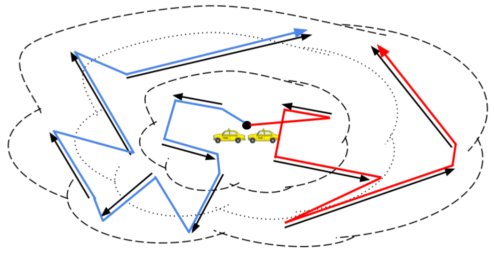


Figure 1: A visualization of the structure of routes computed by our algorithms (e.g., Algorithm 1). The frontiers are dashed and those in thicker dashed lines are selected. The requests are black arrows and routes are colored segmented arrows.

Corollary 3 (Corollary 2.2 in [23]). *The shortest $1 \rightsquigarrow n$ path in $CG(w_1, \ldots, w_n)$ 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 $\{(j, w_j) : j = 1, \ldots, n\}$, and only visits nodes corresponding to extreme points of $(w_1, \ldots, w_n)$.*

4 POINT-TO-POINT REQUESTS

In this section, we present polynomial-time constant-factor approximation algorithms for MLP and single-depot/multi-depot k-MLP with point-to-point requests. The main idea is to first reduce the instance to a point-request instance in a modified metric and, then, solve linear program (LP) and round the fractional optimal solution to nearly optimal routes via the arborescence-packing result and concatenation process described in Section 3. For single-depot k-MLP and MLP, we have a lossless reduction to a point-request instance in a *directed* metric; for multi-depot k-MLP, we reduce to the point-request version in an undirected metric incurring a factor-3 loss.

Intuitively speaking, our algorithms find a sequence of frontiers of increasing sizes around the depot(s) and route the vehicle(s) to satisfy the requests in the order determined by a subsequence of select frontiers. See Figure 1 for a visualization.

Our main results are as follows. We prove the single-depot results in Section 4.1 and the multi-depot ones in Section 4.2.

Theorem 4. *For MLP with point-to-point requests, we can compute a $(\mu^* + \epsilon)$-approximate solution in time $\text{poly}(\mathcal{I}, \frac{1}{\epsilon})$ for any $\epsilon > 0$.*

Theorem 5. *For single-depot k-MLP with point-to-point requests, we can compute a $(2.5\mu^* + \epsilon)$-approximate solution in time $\text{poly}(\mathcal{I}, \frac{1}{\epsilon})$ for any $\epsilon > 0$. ($2.5\mu^* \approx 8.978$.)*

Theorem 6. *For multi-depot k-MLP with point-to-point requests, we can compute a $(25.488+\epsilon)$-approximate solution in time $\text{poly}(\mathcal{I}, \frac{1}{\epsilon})$ for any $\epsilon > 0$.*

4.1 Single Depot

The algorithm leading to Theorem 5 is given in Algorithm 1; the improved ratio for MLP (Theorem 4) is due to a simple observation and the underlying algorithm and analysis are essentially identical. As noted earlier, we transform the given undirected problem instance with point-to-point requests to a point-request instance in a *directed* metric (see Step 1 of Algorithm 1) and apply the linear programming approach in Section 3. The transformation to a directed metric is lossless since there is a one-to-one correspondence between feasible solutions in G and G' and corresponding solutions have the same total latency. Algorithm 1 works with the directed instance G' in Steps 1–7 and with the given undirected instance G in Steps 7 and 8.

The analysis closely resembles the one in [23] but is more involved due to "directionality" of the requests since a vehicle serves a request by going from the source to the destination, not the other way around. Lemma 7 relates the lower-envelope curve f of C in Step 5 and the objective value of the fractional optimal solution in Step 2 as in [23], since this only requires nonnegative edge costs.

Lemma 7. *(i)* $\int_1^n f(x)dx \leq 5\sum_{u\in V, t\in[T]} tx'_{u,t}$. *(ii) If* $(\ell, f(\ell))$ *is a corner point of* f*, then there is a tree* Q^*_ℓ *and time* t^*_ℓ *satisfying the properties stated in Step 5 in Algorithm 1.*

Proof. This is a modified version of Lemma 6.3 in [23] and has essentially the same proof. The only difference is that we add the point $\left(|V(Q^t_\ell) \cap S(t)|, \frac{3c'(Q^t_\ell)}{k} + 2t\right)$ to C. By Theorem 1, $\mathrm{E}[c'(Q^t_\ell)] \leq kt$ and, hence, $\mathrm{E}\left[\frac{3c'(Q^t_\ell)}{k} + 2t\right] \leq 5t$. The rest follows the same proof and the lemma follows. □

Lemma 8. *If* $(\ell, f(\ell))$ *is a corner point of* f*, each of the tours* $Z_{1,\ell}, \ldots, Z_{k,\ell}$ *created in Step 7 has length at most* $\frac{2c'(Q^*_\ell)}{k} + 2t^*_\ell$*. In particular, the part of each tour without the first and last edges connecting to the root has length at most* $\frac{2c'(Q^*_\ell)}{k}$.

Proof. When breaking the cycle Z_ℓ, a break point can be either on a node or on an edge. In the former case, the node appears in two consecutive segments, and in the latter case, the edge is removed. In either cases, all the nodes of Q^*_ℓ will be contained in at least one segment and the corresponding request will be covered in one of the k tours $Z_{1,\ell}, \ldots, Z_{k,\ell}$ by the construction below.

Consider a segment and the corresponding tour $Z_{i,\ell}$. In the cycle Z_ℓ, we visit a node through a "correctly" oriented edge (i.e., in the same direction as the tour) for the first visit and through "incorrectly" oriented edges (i.e., in the opposite direction) for subsequent visits, and consequently, the segment consists of both correctly and incorrectly oriented edges. Without loss of generality, let $R_1, \ldots, R_q$ be the requests corresponding to the vertices that are visited for the first time within the given segment in Z_ℓ. Then, $Z_{i,\ell} = r_0 s_1 d_1 \cdots s_q d_q r_0$. For $P = d_1 s_2 d_2 \cdots s_q d_q$, we show $c(P) \leq \frac{2c'(Q^*_\ell)}{k}$. By the definition of c', each arc in G' corresponds to a path of length 2 in the original graph G. We replace the edges of the segment with corresponding paths of length 2 of the form $d_i s_j d_j$. The resulting path in G has length at most $\frac{2c'(Q^*_\ell)}{k}$ and contains the nodes of P as a subsequence, not necessarily contiguously. We shortcut to get P exactly where shortcutting involves going directly from d_i to s_{i+1} for some i and truncating the beginning or end. Since G has metric edge costs $\{c_e\}$, it follows that $c(P) \leq \frac{2c'(Q^*_\ell)}{k}$.

For the two paths $r_0 s_1 d_1$ and $d_q r_0$ that complete P into $Z_{i,\ell}$, we upper bound the length of each path by t^*_ℓ. Let r be the root in G' and $v_1, v_q \in V'$ be the vertices corresponding to d_1 and d_q, respectively. Since v_1 and v_q are in $Q^*_\ell \cap S(t^*_\ell)$, $c'_{rv_1} \leq t^*_\ell$ and $c'_{rv_q} \leq t^*_\ell$. Then, $c_{r_0 s_1} + c_{s_1 d_1} = c'_{rv_1} \leq t^*_\ell$ and $c_{r_0 d_q} \leq c_{r_0 s_q} + c_{s_q d_q} = c'_{rv_q} \leq t^*_\ell$. It follows that $c(Z_{i,\ell}) = c(P) + c_{r_0 s_1} + c_{s_1 d_1} + c_{r_0 d_q} \leq \frac{2c'(Q^*_\ell)}{k} + 2t^*_\ell$. □

Proof of Theorem 5. We claim that the solution returned by Algorithm 1 has total latency at most the length of P_C in the concatenation graph $CG(f(1), \ldots, f(n))$. By Corollary 3 and Lemma 7 part (i), the length of P_C is at most $\frac{\mu^*}{2}\int_1^n f(x)dx \leq \frac{5\mu^*}{2}\sum_{v,t} tx'_{v,t} = 2.5\mu^*\sum_{v,i,t} tx^i_{v,t}$. Also, $\sum_{v,i,t} tx^i_{v,t}$ is at most $(1+\epsilon)$ times the optimal latency, where the $(1+\epsilon)$-factor is due to scaling and rounding.

We now prove the claim. More specifically, we show that the total latency is at most the length of P_C in $CG(f(1), \ldots, f(n))$ by induction. By Theorem 2 and Lemma 7, there exist Q^*_ℓ and t^*_ℓ satisfying the properties stated in Step 5 for each corner point $(\ell, f(\ell))$ on P_C. By Lemma 8, the tours that would be created from Q^*_ℓ have length at most $\frac{2c'(Q^*_\ell)}{k} + 2t^*_\ell$ in the correct direction and cover all the requests corresponding to the vertices of Q^*_ℓ. In particular, the part of each tour without the first and last edges connecting to the root has length at most $\frac{2c'(Q^*_\ell)}{k}$ in the correct direction.

Suppose inductively that we have covered at least o requests by the partial solution constructed by concatenating tours corresponding to the nodes on P_C up to and including o. Assume we next take an edge (o, ℓ) and consider the additional contribution to the total latency when we concatenate $Z_{i,\ell}$ to vehicle i's route for $i \in [k]$. Note the resulting partial solution covers at least ℓ requests.

Each request covered in the current concatenation step incurs additional latency of at most $\frac{f(\ell)}{2}$ in expectation. To see this, let L be the length of the part of a tour $Z_{i,\ell}$ without the root r_0 and L' be the length up to the destination of a request covered by the tour. With probability $\frac{3}{4}$, the additional latency incurred for the request is $t_\ell + L'$. With probability $\frac{1}{4}$, the additional latency is at most $t_\ell + 3(L - L')$ because the length of a traversal in the opposite direction is upper bounded by 3 times the length of the corresponding traversal in the correct direction (accounting the s_i-d_i portions 3 times). For example, if $s_1 d_1 \ldots s_n d_n$ is a traversal in the correct direction, the reverse traversal is a shortcut version of $d_n s_n d_n s_n \ldots d_1 s_1 d_1 s_1$ which has length at most 3 times that of $s_1 d_1 \ldots s_n d_n$. In expectation, the additional latency incurred for the request is at most $\frac{3}{4}(t_\ell + L') + \frac{1}{4}(t_\ell + 3L - 3L') = t_\ell + \frac{3L}{4} \leq \frac{f(\ell)}{2}$ as $L \leq \frac{2c'(Q^*_\ell)}{k}$.

By a similar argument, each request that is still uncovered after concatenation incurs additional latency of at most $f(\ell)$ in expectation. The traversal in the correct direction has length at most $2t_\ell + L$ and the one in the opposite direction has length at most $2t_\ell + 3L$. In expectation, the additional latency is at most $2t_\ell + \frac{3L}{2} \leq f(\ell)$.

There are exactly ℓ requests covered by the tours $Z_{1,\ell}, \ldots, Z_{k,\ell}$ and at most $\ell - o$ new requests are covered in the current concatenation step. Each of these incurs additional latency at most $\frac{f(\ell)}{2}$. At most $n - \ell$ requests remain to be covered and they each incur additional latency at most $f(\ell)$. The overall increase in the total latency is at most $\frac{f(\ell)}{2}(\ell - o) + f(\ell)(n - \ell) = f(\ell)\left(n - \frac{o+\ell}{2}\right)$ which

Algorithm 1. *The input is a single-depot k-MLP instance $G = (V, E)$ with point-to-point requests and root r_0.*

```
1. Define a directed graph $G' = (V', A)$ on $n + 1$ nodes corresponding to the root/requests and arcs $(v_i, v_j)$ with length $c'_{v_i v_j} = c_{d_i s_j} + c_{s_j d_j}$ for all requests $i$ and $j$. Treat root $r_0$ as the 0-th request $(s_0, d_0)$ with $r_0 = s_0 = d_0$. Let $r$ denote the root in $G'$. Let $\mathsf{T} = 4n\mathsf{LB}$ where $\mathsf{LB} := \max_{v \in V} c_{r_0 v}$. By scaling and rounding the $c$-metric (e.g., [6]), we may assume that $\mathsf{T} = \mathrm{poly}\big(\mathcal{I}, \frac{1}{\epsilon}\big)$ losing a $(1 + \epsilon)$-factor.
2. Compute an optimal solution $(x, z)$ to (LP) for the instance given by $G'$ and $\{c'_a\}$. Let $x'_{v,t} = \sum_i x^i_{v,t}$, $z'_{a,t} = \sum_i z^i_{a,t'}$ for all $v$, $a$, $t$.
3. Initialize $C \leftarrow \{(1, 0)\}$, $Q \leftarrow \emptyset$. 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 $t > 0$.)
4. For all $t = 1, \dots, \mathsf{T}$, do the following. Apply Theorem 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 $(\gamma_1, Q^t_1), \dots, (\gamma_q, Q^t_q)$. For each arboresence $Q^t_\ell$ in the family, add the point $\big(|V(Q^t_\ell) \cap S(t)|, \frac{3c'(Q^t_\ell)}{k} + 2t\big)$ to $C$, and add $Q^t_\ell$ to $Q$.
5. For all $\ell = 1, \dots, n$, compute $f(\ell)$ where $f : [1, n] \mapsto \mathbb{R}_+$ is the lower-envelope curve of $C$. We show in Lemma 7 that for every corner point $\big(\ell, f(\ell)\big)$ of $f$, there is some tree $Q^*_\ell \in Q$ and some time $t^*_\ell$ such that $\ell = |V(Q^*_\ell) \cap S(t^*_\ell)|$, $f(\ell) = \frac{3c'(Q^*_\ell)}{k} + 2t^*_\ell$, and $\max_{v \in Q^*_\ell \cap S(t^*_\ell)} c'_{rv} \le t^*_\ell$.
6. Find a shortest $1 \rightsquigarrow n$ path $P_C$ in the concatenation graph $CG(f(1), \dots, f(n))$.
7. For every node $\ell > 1$ on $P_C$, do the following. Do a DFS-traversal from $r$ on $Q^*_\ell$ to obtain a cycle $Z_\ell$, in the undirected sense, that uses each arc of $Q^*_\ell$ twice. Break $Z_\ell$ into $k$ segments, each of $c'$-length at most $2c'(Q^*_\ell)/k$. For each segment, create the corresponding tour $Z_{i,\ell}$ in $G$, starting from and ending at $r_0$, that satisfies only the requests that have their (first) appearance within the segment in the Eulerian tour and in the order of first appearances. Skip requests that are not in $S(t^*_\ell)$. With probability $\frac{3}{4}$, we traverse $Z_{i,\ell}$ in the correct direction and with probability $\frac{1}{4}$, in the opposite direction (satisfying the same set of requests but in the reverse order). This yields a collection of $k$ tours $Z_{1,\ell}, \dots, Z_{k,\ell}$.
8. For every $i = 1, \dots, k$, concatenate the tours $Z_{i,\ell}$ for nodes $\ell$ on $P_C$ to obtain vehicle $i$'s route, shortcutting if a request has already been satisfied.
```

is equal to the length of the edge (o, ℓ) in $CG(f(1), \dots, f(n))$. By induction, the total latency is at most the length of P_C.

For the running time, since $\mathsf{T} = \mathrm{poly}(\mathcal{I}, \frac{1}{\epsilon})$, we can design a separation oracle for (3), which is a min-cut algorithm, and solve (LP) in time $\mathrm{poly}(\mathcal{I}, \frac{1}{\epsilon})$. □

Proof of Theorem 4. We follow the same analysis above for Algorithm 1. The improvement of the approximation ratio comes from the fact that we do not need to break the cycle Z_ℓ into k tours in Step 7 since we have exactly one vehicle. Moreover, we note that the length of a traversal of Z_ℓ in *either* direction is at most $2c'(Q^*_\ell)$. In Step 4, we now add the point $\big(|V(Q^t_\ell) \cap S(t)|, 2c'(Q^t_\ell)\big)$ to C to leverage this fact. Following the proof of Theorem 5, we now therefore obtain a μ^*-approximation. □

We remark that for single-depot k-MLP, we can avoid the additive ϵ in the approximation factors by utilizing the more combinatorial approach presented in [23]. In this approach, we obtain the arborescences directly without solving an LP. Specifically, for each $i = 1, \dots, n$, we aim to find the least c'-cost arborescence spanning at least i requests (which in turn utilizes arborescence packings). We then again use the concatenation graph to select a subset that will be converted into tours which are then concatenated. This is closer to the approach we follow in our experimental results.

4.2 Multiple Depots

For the more general multi-depot k-MLP with point-to-point requests, we provide a reduction showing that an α-approximation algorithm for the point-request version on undirected metrics yields a 3α-approximation algorithm for the point-to-point request version on undirected metrics. More specifically, the following result immediately yields Theorem 6 using the $(8.496 + \epsilon)$-approximation for multi-depot k-MLP with point-requests in [23]. The proofs in the remainder of this section are deferred to the full version of the paper [12].

Lemma 9. *Let OPT and OPT′ denote respectively the optimal values of the multi-depot k-MLP instance with point-to-point requests, and the multi-depot k-MLP instance with point requests obtained by the reduction under the c' edge costs (described below). Then $OPT \le OPT' \le 3 \cdot OPT$. Hence, an α-approximate solution to the point-requests instance yields a 3α-approximate solution to the point-to-point requests instance.*

The main idea of the factor-3 reduction is to identify the requests by their source nodes and incorporate the request-specific distances $c_{s_i d_i}$ into distances between sources in a symmetric way. We treat the root nodes as dummy requests with the identical source and destination nodes for the reduction. More specifically, G' contains only root nodes and nodes representing the requests, and the distance between v_i and v_j in G' is $c'_{v_i v_j} = c_{s_i s_j} + c_{s_i d_i} + c_{s_j d_j}$. It is easy to see that (G', c') is a metric. We then solve the resulting problem instance using an algorithm for the point requests and convert the computed paths into ones in the original problem instance. Via this reduction, observe that the latency of a rooted path in the (G', c') instance corresponds to the latency of a particular type of path, that we call a *backtracking path*, in the original problem instance, where a vehicle satisfies each request i by going from s_i to d_i and then returning to s_i before moving onto the next request. The backtracking paths are for the purpose of analysis only and we shortcut these paths to obtain the final routes for the vehicles. Clearly, there are bijective mappings between request-orderings and paths and backtracking paths. If the order of the requests is $R_1 \cdots R_n$, which equivalently corresponds to a path P, the corresponding backtracking path is

$$P_b = r_0 s_1 d_1 s_1 s_2 d_2 s_2 \cdots s_n d_n \,.$$

If the order of the requests is $R_1 \cdots R_n$ on the route P of a vehicle starting from, say, root r_0, the latency of the i-th request on the corresponding backtracking path P_b is determined as

$$\mathrm{Lat}(P_b, i) = \begin{cases} \mathrm{Lat}(P_b, i-1) + c_{d_{i-1} s_{i-1}} + c_{s_{i-1} s_i} + c_{s_i d_i}, & i > 1 \\ 0, & i = 0 \end{cases} .$$

The total latency of P_b is $\mathrm{Lat}(P_b) = \sum_{i=1}^{n} \mathrm{Lat}(P_b, i)$. In particular, the following property is useful in proving Lemma 9.

Lemma 10. *For any path P and corresponding backtracking path P_b, we have $\mathrm{Lat}(P) \leq \mathrm{Lat}(P_b) \leq 3\,\mathrm{Lat}(P)$. Further, the factor of 3 is tight.*

5 RELEASE-TIME CONSTRAINTS

We show constant-factor approximation algorithms for the minimum latency problems with release-time constraints for both point requests and point-to-point requests. We incorporate the release-time constraints into the linear program (LP) and solve the resulting linear program optimally as before. When rounding an optimal fractional solution to an integral solution, we follow the same analysis steps in Section 4 but need to account for the release times. Our analysis can be easily modified to satisfy the release-time constraints with little or no extra cost in terms of approximation guarantees.

For point requests, we have the following results with similar approximation ratios as for the variants without release times. We prove them in Sections 5.1 and 5.2, respectively:

Theorem 11. *For point requests with release times, we obtain the following approximation guarantees in time $\mathrm{poly}(\mathcal{I}, \frac{1}{\epsilon})$ for any $\epsilon > 0$:*

(1) A $(2\mu^* + \epsilon)$-approximation for MLP and single-depot k-MLP; ($2\mu^* \approx 7.183$)
(2) A $(13.728 + \epsilon)$-approximation for multi-depot k-MLP.

For point-to-point requests, we also have constant-factor approximation algorithms:

Theorem 12. *For point-to-point requests with release times, we have the following approximation guarantees in time $\mathrm{poly}(\mathcal{I}, \frac{1}{\epsilon})$ for any $\epsilon > 0$:*

(1) A $(2\mu^* + \epsilon)$-approximation for MLP; ($2\mu^* \approx 7.183$)
(2) A $(2.5\mu^* + \epsilon)$-approximation for single-depot k-MLP; ($2.5\mu^* \approx 8.978$)
(3) A $(41.184 + \epsilon)$-approximation for multi-depot k-MLP.

As in the case without release times, the total latency objective is computed with respect to the beginning of the paths at the root nodes. This is analogous to the total-completion-time objective that is widely used in the scheduling literature (see, e.g., [31]). While, admittedly, measuring the latency of a request R_i by (time taken to reach d_i) $-T_i$, which accounts for the actual time R_i spends in the system, yields a more natural objective that is akin to the *flow-time* objective in scheduling, the resulting optimization problem is much harder to approximate (as is the case in scheduling). Therefore, following much of the work in scheduling, we consider our latency objective, which is a reasonable first step in studying ride-sharing and delivery problems with release times.

To distinguish from the latency objective without release times, we use Lat^+ to denote latencies and incorporate the release times by making vehicles wait at the source of a request if it has not been released. For example, for path $P = r_0 s_1 d_1 \cdots s_n d_n$, note that the latency of the i-th request on P is

$$\mathrm{Lat}^+(P, i) = \begin{cases} \max\{\mathrm{Lat}^+(P, i-1) + c_{d_{i-1}s_i}, T_i\} + c_{s_i d_i}, & i > 1 \\ 0, & i = 0 \end{cases}.$$

Similarly, the length of a path/tour needs to correspond to the traversal time subject to release-time constraints. Given a sequence of requests to be satisfied, the traversal time for the sequence consists of *driving times* for moving from one location to another and *waiting times* for staying fixed at a location before a request is released. We can still concatenate and shortcut in the sense that the total traversal time of concatenation of two paths meeting at a node is at most the sum of traversal times of the paths, and the traversal time of a shortcut path is at most that of the original path. The algorithms and their analyses stay essentially the same for traversal times as for lengths of paths/tours.

5.1 Point Requests: Proof of Theorem 11

Part (1). We incorporate the following release-time constraints into the linear program (LP) and solve for an optimal fractional solution as before:

$$x^i_{v,t} = 0 \text{ if } T_i > t \qquad \forall v, t, i\,. \tag{7}$$

(Since we have point requests, the c' edge costs used in Algorithm 1 are simply the bidirected c costs.) We now need to incorporate the R_i's in our upper bound T, but $\log \mathsf{T}$ is still polynomially bounded and we can obtain a $(1 + \epsilon)$-approximate solution to the LP. Then, we round the fractional solution as in Algorithm 1 by creating a set of tours starting and ending at the root and concatenating them.

In the analysis, we upper bound the traversal times of the tours using the same upper bound (or nearly the same) we have used for the lengths of the tours, and, hence, obtain the same or similar approximation guarantees with respect to the total latency objective as for the variants without release times. Specifically, Theorem 6.1 in Post and Swamy [23], which is the equivalent of Theorem 5 for point requests with release times, obtains an approximation ratio of $2\mu^*$ for single-depot k-MLP with point requests without release times. We observe that the same upper bound of $\frac{2c(Q^*_\ell)}{k} + 2t^*_\ell$ on the c-length of a tour obtained for time-point t^*_ℓ used in Theorem 6.1 in [23] also upper bounds the traversal time of any tour $Z_{i,\ell}$ that we obtain in Step 7 of Algorithm 1. This is because the upper bound of t^*_ℓ on the lengths of the edges connecting the ends of $Z_{i,\ell}$ to the root also upper bounds the waiting time portion of the traversal: since all requests on $Z_{i,\ell}$ have release time at most t^*_ℓ by the release-time constraints (7), t^*_ℓ also accounts for the total waiting time of the traversal of $Z_{i,\ell}$. Thus, we do not incur any extra cost; the rest of the analysis is the same as that in [23], and the approximation guarantee follows.

Part (2). For this result, we use Algorithm 1 in [23] and its analysis, which uses a related but different linear program. We incorporate the release-time constraints (7) as before. Due to the space constraints, we omit the linear program and refer to [23] for details. As before, we upper bound the traversal times of tours obtained for some time-point t by (the total driving time of at most t to visit the requests in the tour) + t, since t is an upper bound on the waiting time incurred as all requests on the tour have release times at most t. Consequently, following the analysis in [23], one can argue that for any constant $1 < c < e$, we obtain an approximation ratio of at most $\frac{(2c+1)(1-e^{-1})}{\ln c(1-ce^{-1})} + \epsilon$.[4] Taking $c = 1.58726$, we obtain an approximation ratio of at most $13.7272 + \epsilon$.

5.2 Point-to-Point Requests: Proof of Theorem 12

The proof follows the same reasoning as in the proof of Theorem 11. For parts (1) and (2), we move to a directed metric c' as in Step 1 of Algorithm 1, incorporate release times as in (7), solve the resulting LP (LP), and round it using Algorithm 1. The guarantees in parts (1) and (2) rely on the analysis in Theorems 4 and 5 respectively, and the now-familiar fact that the waiting time for a tour for time t is at most t.

For part (3), we use the following factor-3 reduction to the point-requests and use part (2) of Theorem 11. This yields a $(41.184 + \epsilon)$-approximation. The reduction is analogous to the constant-factor reduction presented in Section 4.2, but proof details are more involved and deferred to the full version [12]. For path $P = r_0 s_1 d_1 \cdots s_n d_n$, we have the following relations for the latencies of the corresponding backtracking path P_b,

$$\text{Lat}^+(P_b, i) = \begin{cases} \max\left\{ \begin{array}{l} \text{Lat}^+(P_b, i-1) \\ + c_{d_{i-1}s_{i-1}} + c_{s_{i-1}s_i} \end{array}, T_i \right\} + c_{s_i d_i}, & i > 1 \\ 0, & i = 0 \end{cases}.$$

We have the following property of the backtracking paths with release times and, hence, the approximation guarantee via the reduction.

Lemma 13. *For any path P and corresponding backtracking path P_b, we have* $\text{Lat}^+(P) \le \text{Lat}^+(P_b) \le 3\,\text{Lat}^+(P)$.

Lemma 14. *Given point-to-point requests with release times, the optimal backtracking path has total latency at most 3 times that of the optimal path. More generally, an α-approximate backtracking path has total latency at most 3α times that of the optimal path.*

6 EXPERIMENTS

In this section, we will present a comparison of our algorithm and a natural greedy baseline on two public datasets, *viz.*, the Chicago taxi data[5] and the NYC taxi data[6]. We focus on the single depot k-MLP problem with release times; we also evaluated our algorithms for the k-MLP problem without release times on the same datasets and obtained qualitatively similar results.

6.1 Datasets

The Chicago dataset consists of 27 million taxi trips from 2013 to 2016, consisting of pick-up times, drop-off times, pick-up locations, drop-off locations, trip times and trip lengths. The pick-up and drop-off locations are specified in terms of Census Tracts instead of precise latitude and longitude values. For our analysis and graph construction, we cover the city using equal-sized geographical cells of size 1000-sq. meters, and map the Census Tracts to these cells. Each taxi trip is then treated as a trip between two cells. We assign pairwise distances between each pair of cells to be the driving time between centroids of each cell, computed using Google Maps API.

The New York taxi dataset consists of more than 1 billion taxi trips from 2009 to 2015 with location information. We discretize the pick-up and drop-off locations of each trip to their respective geographical cells, and compute pairwise distances as before.

We randomly choose one weekday for both datasets, 7/17/2015 for Chicago and 5/31/2013 for New York, and consider all taxi rides in a one-hour (5PM to 6PM) and two-hour (5PM to 7PM) window. We set the release-times to be equal to the pick-up times of the requests. We also choose a random cell in each city as the depot location. We refer the reader to the full paper version [12] for more details on the datasets.

6.2 Algorithm Implementations

We first describe the natural greedy algorithm for the problem, that we use as a baseline. For each taxi that becomes idle, the greedy algorithm scans through all unassigned trip requests, and assigns the trip request with the smallest resulting finish-time. The resulting solution has at most k paths branching out from the root and we refer to such solution as a *k-path solution*. A path of a k-path solution will correspond to the path of a taxi.

Even though our Algorithm 1 from Section 4.1 has constant approximation guarantees, it is relatively complex and hard to implement in practice. We therefore use the following practical heuristic (that we call kMLP-Fast) based on Algorithm 1 (in the sequel, n refers to the number of trip requests and k refers to the number of taxis):

1. Sort the requests in increasing order of their release times
2. Define a layer(i) solution as the greedy k-path solution considering only the first 2^i requests. Generate layers for $i \in [1, 2, \ldots, \log(n)]$ where each layer i consists of at most k paths of the layer(i) solution.
3. Create a "concatenation graph" where each node corresponds to a path in a given layer. We create an edge from path P_i in layer i to path P_j in layer j for $j > i$, and set the edge cost as follows: $e(P_i, P_j) = \sum_{t \in P_j \setminus P_i} (\text{Lat}^+(P_i + P_j, t) - \text{Lat}^+(P_j, t)) + (n - 2^j) \cdot \text{length}(P_j)$, where t indexes the requests in P_j that are not in P_i, $P_i + P_j$ represents the path obtained by appending P_j to P_i and removing duplicates, and length(P_j) is the traversal time of the path (which includes both waiting and driving times). Essentially, the edge cost is an upper bound on the additional latency incurred by trips in P_j and all subsequent trips in higher layers, when appending P_j to P_i. Create a dummy start node s corresponding to an empty path, and connect every node to s with edge costs computed as before. Create a finish node t and connect every path in the last layer ($i = \log(n)$) to t, with edge cost 0.
4. Assign a capacity of 1 to each edge in the concatenation graph, and run a min-cost flow algorithm from s to t with the target flow amount k. Concatenate the paths traversed by each flow (removing duplicates) to generate the final solution.

The main simplification that the kMLP-Fast algorithm performs (compared to Algorithm 1) is that it greedily defines requests for each layer by sorting them by their release times. For each layer,

[4]Claim 5.2 (iii) in [23] changes as follows: conditioned on the random offset h, the expected latency of a node v first covered in iteration j of the algorithm is now at most $(1+\epsilon)(3t_0 + 3t_1 + \ldots + 3t_{j-1} + 2t_j) \le \frac{(1+\epsilon)(2c+1)}{c-1} \cdot t_j$

[5]http://digital.cityofchicago.org/index.php/chicago-taxi-data-released/

[6]http://www.nyc.gov/html/tlc/html/about/trip_record_data.shtml

it then creates a k-path solution using the greedy-heuristic as a subroutine to cover all nodes in that layer. This differs from the paths obtained in Algorithm 1. Finally, the algorithm concatenates the paths across different layers using a min-cost flow algorithm (as opposed to a shortest path in the concatenation-graph, since we now work with at most k paths per layer.)

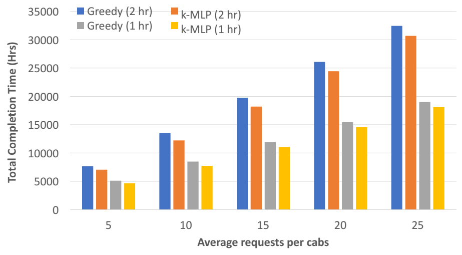
6.3 Latency Objective

We compare the kMLP-Fast and greedy algorithms on the total-latency objective, which includes both the waiting time and the driving time for a request. Figure 2 illustrates the performance of the algorithms on both datasets. Note that on the y-axis, we report values representing the *sum* of the corresponding metric over all requests in the observed interval.

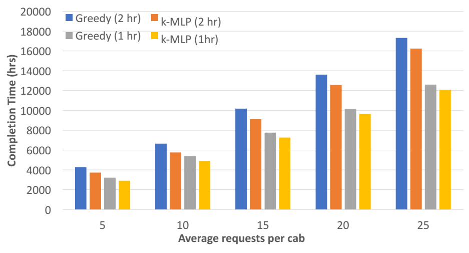
One trend we observe in both cities is that as the number of taxis increases, our algorithm outperforms the greedy algorithm by a wider margin. Indeed, in New York City, our algorithm does better by around 5% - 8% as the average number of trips for a taxi decreases from 25 to 5. A similar trend, with the difference ranging from 6% to 13%, is observed in Chicago. The reason for the higher difference in Chicago compared to New York City might be attributed to the different demand distributions in the two cities. The requests in New York are more densely concentrated in a smaller region, which helps the greedy algorithm.

6.4 Other Desiderata

While the user satisfaction in terms of latency is a good characteristic of a ride-sharing service, other characteristics such as the total distance covered by the taxis in the system, the efficiency of the taxis in terms of taxi idle times, the load on each taxi, etc are also good properties of any ride-sharing service. We measured the

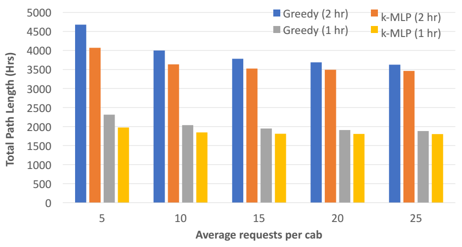


(a) New York City

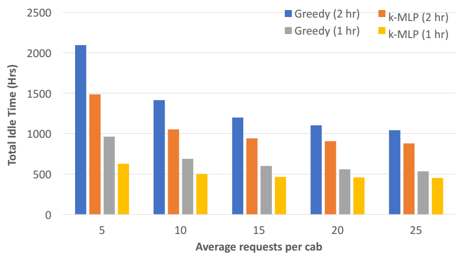


(b) Chicago

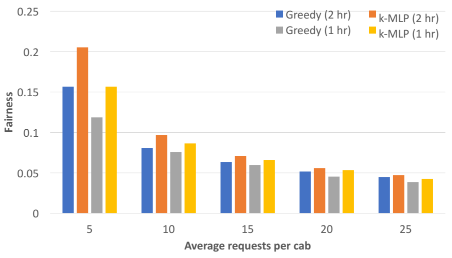
Figure 2: Total latency of all trips in hours



(a) New York City - total length in hours



(b) New York City - total idle time of cabs in hours



(c) New York City - balanced allocations

Figure 3: The performance of kMLP-Fast algorithm w.r.t. other natural measures.

performance of the kMLP-Fast algorithm with respect to these metrics. Figure 3 illustrates the relative performance of our algorithm compared to the greedy baseline for the New York City dataset. We also obtain similar performances for the Chicago dataset and provide these results in the full paper version [12].

Even though the kMLP-Fast algorithm is optimized for the latency objective, it performs surprisingly well in minimizing the total distance a taxi travels to serve its rides. In fact, the cabs travel between 5% and 15% less on this measure. Another promising result is the performance on the measure of idle time a taxi spends transitioning between rides. Here again, our algorithm performs significantly better by 15% to 35% over the greedy baseline. Finally, we measured the fairness as the coefficient of variation of the total trip lengths of taxis. A fair allocation ensures taxis are generally equally loaded, resulting in a smaller value of this measure. On this measure, the greedy algorithm underperforms in New York City while the difference is much less in Chicago. Note that the greedy algorithm tends to produce more balanced paths by its design.

REFERENCES

[1] Hernán Abeledo, Ricardo Fukasawa, Artur Pessoa, and Eduardo Uchoa. 2013. The time dependent traveling salesman problem: polyhedra and algorithm. *Mathematical Programming Computation* 5, 1 (01 Mar 2013), 27–55. https://doi.org/10.1007/s12532-012-0047-y

[2] Aamena Alshamsi, Sherief Abdallah, and Iyad Rahwan. 2009. Multiagent Self-organization for a Taxi Dispatch System. In *Proceedings of the 8th International Conference on Autonomous Agents and Multiagent Systems (AAMAS '09)*.

[3] F. Angel-Bello, Y. Cardona-Valdés, and A. Álvarez. 2017. Mixed integer formulations for the multiple minimum latency problem. *Operational Research* (08 Feb 2017). https://doi.org/10.1007/s12351-017-0299-4

[4] Aaron Archer and Anna Blasiak. 2010. Improved Approximation Algorithms for the Minimum Latency Problem via Prize-collecting Strolls. In *Proceedings of the Twenty-first Annual ACM-SIAM Symposium on Discrete Algorithms (SODA '10)*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 429–447. http://dl.acm.org/citation.cfm?id=1873601.1873637

[5] Aaron Archer, Asaf Levin, and David P. Williamson. 2008. A Faster, Better Approximation Algorithm for the Minimum Latency Problem. *SIAM J. Comput.* 37, 5 (2008), 1472–1498. https://doi.org/10.1137/07068151X arXiv:https://doi.org/10.1137/07068151X

[6] Sanjeev Arora and George Karakostas. 2003. Approximation Schemes for Minimum Latency Problems. *SIAM J. Comput.* 32, 5 (2003), 1317–1337. https://doi.org/10.1137/S0097539701399654 arXiv:https://doi.org/10.1137/S0097539701399654

[7] Giorgio Ausiello, Stefano Leonardi, and Alberto Marchetti-Spaccamela. 2000. *On Salesmen, Repairmen, Spiders, and Other Traveling Agents*. Springer Berlin Heidelberg, Berlin, Heidelberg, 1–16.

[8] Avrim Blum, Prasad Chalasani, Don Coppersmith, Bill Pulleyblank, Prabhakar Raghavan, and Madhu Sudan. 1994. The Minimum Latency Problem. In *Proceedings of the Twenty-sixth Annual ACM Symposium on Theory of Computing (STOC '94)*. ACM, New York, NY, USA, 163–171. https://doi.org/10.1145/195058.195125

[9] Kamalika Chaudhuri, Brighten Godfrey, Satish Rao, and Kunal Talwar. 2003. Paths, Trees, and Minimum Latency Tours. In *Proceedings of the 44th Annual IEEE Symposium on Foundations of Computer Science (FOCS '03)*. IEEE Computer Society, Washington, DC, USA, 36–. http://dl.acm.org/citation.cfm?id=946243.946316

[10] Chandra Chekuri, Nitish Korula, and Martin Pál. 2012. Improved Algorithms for Orienteering and Related Problems. *ACM Trans. Algorithms* 8, 3, Article 23 (July 2012), 27 pages. https://doi.org/10.1145/2229163.2229167

[11] Chandra Chekuri and Amit Kumar. 2004. *Maximum Coverage Problem with Group Budget Constraints and Applications*. Springer Berlin Heidelberg, Berlin, Heidelberg, 72–83.

[12] Abhimanyu Das, Sreenivas Gollapudi, Anthony Kim, Debmalya Panigrahi, and Chaitanya Swamy. 2018. Minimizing Latency in Online Ride and Delivery Services. (Feb 2018). Available at https://arxiv.org/abs/1802.02744.

[13] Willem E. de Paepe, Jan Karel Lenstra, Jiri Sgall, RenÃľ A. Sitters, and Leen Stougie. 2004. Computer-Aided Complexity Classification of Dial-a-Ride Problems. *INFORMS Journal on Computing* 16, 2 (2004), 120–132. https://doi.org/10.1287/ijoc.1030.0052 arXiv:https://doi.org/10.1287/ijoc.1030.0052

[14] Jittat Fakcharoenphol, Chris Harrelson, and Satish Rao. 2003. The K-traveling Repairman Problem. In *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA '03)*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 655–664. http://dl.acm.org/citation.cfm?id=644108.644215

[15] Michel Goemans and Jon Kleinberg. 1996. An Improved Approximation Ratio for the Minimum Latency Problem. In *Proceedings of the Seventh Annual ACM-SIAM Symposium on Discrete Algorithms (SODA '96)*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 152–158. http://dl.acm.org/citation.cfm?id=313852.313909

[16] R.L. Graham, E.L. Lawler, J.K. Lenstra, and A.H.G.Rinnooy Kan. 1979. Optimization and Approximation in Deterministic Sequencing and Scheduling: a Survey. In *Discrete Optimization II*, P.L. Hammer, E.L. Johnson, and B.H. Korte (Eds.). Annals of Discrete Mathematics, Vol. 5. Elsevier, 287 – 326. https://doi.org/10.1016/S0167-5060(08)70356-X

[17] Elias Koutsoupias, Christos Papadimitriou, and Mihalis Yannakakis. 1996. *Searching a fixed graph*. Springer Berlin Heidelberg, Berlin, Heidelberg, 280–289.

[18] Zhixing Luo, Hu Qin, and Andrew Lim. 2014. Branch-and-price-and-cut for the multiple traveling repairman problem with distance constraints. *European Journal of Operational Research* 234, 1 (2014), 49–60. https://EconPapers.repec.org/RePEc:eee:ejores:v:234:y:2014:i:1:p:49-60

[19] Isabel Méndez-Díaz, Paula Zabala, and Abilio Lucena. 2008. A New Formulation for the Traveling Deliveryman Problem. *Discrete Appl. Math.* 156, 17 (Oct. 2008), 3223–3237. https://doi.org/10.1016/j.dam.2008.05.009

[20] Nenad Mladenović, Dragan Urošević, and Saïd Hanafi. 2013. Variable neighborhood search for the travelling deliveryman problem. *4OR* 11, 1 (01 Mar 2013), 57–73. https://doi.org/10.1007/s10288-012-0212-1

[21] Samuel Nucamendi-Guillén, Iris Martínez-Salazar, Francisco Angel-Bello, and J Marcos Moreno-Vega. 2016. A mixed integer formulation and an efficient metaheuristic procedure for the k-Travelling Repairmen Problem. *Journal of the Operational Research Society* 67, 8 (01 Aug 2016), 1121–1134. https://doi.org/10.1057/jors.2015.113

[22] Christos H. Papadimitriou and Mihalis Yannakakis. 1993. The Traveling Salesman Problem with Distances One and Two. *Math. Oper. Res.* 18, 1 (Feb. 1993), 1–11. https://doi.org/10.1287/moor.18.1.1

[23] Ian Post and Chaitanya Swamy. 2015. Linear Programming-based Approximation Algorithms for Multi-vehicle Minimum Latency Problems. In *Proceedings of the Twenty-sixth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA '15)*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 512–531. http://dl.acm.org/citation.cfm?id=2722129.2722164

[24] Shiyou Qian, Jian Cao, Frédéric Le Mouël, Issam Sahel, and Minglu Li. 2015. SCRAM: A Sharing Considered Route Assignment Mechanism for Fair Taxi Route Recommendations. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '15)*. ACM, New York, NY, USA, 955–964. https://doi.org/10.1145/2783258.2783261

[25] Meng Qu, Hengshu Zhu, Junming Liu, Guannan Liu, and Hui Xiong. 2014. A Cost-effective Recommender System for Taxi Drivers. In *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '14)*. ACM, New York, NY, USA, 45–54. https://doi.org/10.1145/2623330.2623668

[26] Sartaj Sahni and Teofilo Gonzalez. 1976. P-Complete Approximation Problems. *J. ACM* 23, 3 (July 1976), 555–565. https://doi.org/10.1145/321958.321975

[27] Douglas O. Santos and Eduardo C. Xavier. 2013. Dynamic Taxi and Ridesharing: A Framework and Heuristics for the Optimization Problem. In *Proceedings of the Twenty-Third International Joint Conference on Artificial Intelligence (IJCAI '13)*. AAAI Press, 2885–2891. http://dl.acm.org/citation.cfm?id=2540128.2540544

[28] Marcos Melo Silva, Anand Subramanian, Thibaut Vidal, and Luiz Satoru Ochi. 2012. A simple and effective metaheuristic for the Minimum Latency Problem. *European Journal of Operational Research* 221, 3 (2012), 513 – 520. https://doi.org/10.1016/j.ejor.2012.03.044

[29] René Sitters. 2002. The Minimum Latency Problem Is NP-Hard for Weighted Trees. In *Proceedings of the 9th International IPCO Conference on Integer Programming and Combinatorial Optimization*. Springer-Verlag, London, UK, UK, 230–239. http://dl.acm.org/citation.cfm?id=645591.660083

[30] René Sitters. 2014. Polynomial Time Approximation Schemes for the Traveling Repairman and Other Minimum Latency Problems. In *Proceedings of the Twenty-fifth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA '14)*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 604–616. http://dl.acm.org/citation.cfm?id=2634074.2634120

[31] M. Skutella. 2006. List Scheduling in Order of α-Points on a Single Machine. In *Efficient Approximation and Online Algorithms*. Springer, 250–291.

[32] John N. Tsitsiklis. 1992. Special cases of traveling salesman and repairman problems with time windows. *Networks* 22, 3 (1992), 263–282. https://doi.org/10.1002/net.3230220305

[33] Xianyuan Zhan, Xinwu Qian, and Satish V. Ukkusuri. 2014. Measuring the Efficiency of Urban Taxi Service System. In *The Third International Workshop on Urban Computing (UrbComp '14)*.

[34] Xudong Zheng, Xiao Liang, and Ke Xu. 2012. Where to Wait for a Taxi?. In *Proceedings of the ACM SIGKDD International Workshop on Urban Computing (UrbComp '12)*. ACM, New York, NY, USA, 149–156. https://doi.org/10.1145/2346496.2346520

[35] Chenguang Zhu and Balaji Prabhakar. 2017. Reducing Inefficiencies in Taxi Systems. In *Proceedings of the Fifty-Sixth IEEE Conference on Decision and Control (CDC '17)*.