Last-Mile Delivery Made Practical: An Efficient Route Planning Framework with Theoretical Guarantees

Yuxiang Zeng [†] Yongxin Tong [‡] Lei Chen [†]
[†] The Hong Kong University of Science and Technology, Hong Kong SAR, China
[‡] BDBC, SKLSDE Lab and IRI, Beihang University, China
[†] {yzengal, leichen}@cse.ust.hk [‡] yxtong@buaa.edu.cn

ABSTRACT

Last-mile delivery (LMD) refers to the movement of goods from transportation origins to the final destinations. It has widespread applications such as urban logistics, e-commerce, etc. One fundamental problem in last-mile delivery is route planning, which schedules multiple couriers' routes, *i.e.*, sequences of origins and destinations of the requests under certain optimization objectives. Prior studies usually designed heuristic solutions to two strongly NP-hard optimization objectives: minimizing the makespan (*i.e.*, maximum travel time) of couriers and total latency (*i.e.*, waiting time) of requesters. There is no algorithm with theoretical guarantees for either optimization objective in practical cases. In this paper, we propose a theoretically guaranteed solution framework for both objectives. It achieves both approximation ratios of 6ρ, where ρ is the approximation ratio of a core operation, called kLMD, which plans for one courier a route consisting of k requests. Leveraging a spatial index called hierarchically separated tree, we further design an efficient approximation algorithm for kLMD with $\rho = O(\log n)$, where n is the number of requests. Experimental results show that our approach outperforms state-of-the-art methods by averagely 48.4%-96.0% and 49.7%-96.1% for both objectives. Especially in large-scale real datasets, our algorithm has 29.3×-108.9× shorter makespan and 20.2×-175.1× lower total latency than the state-of-the-art algorithms.

PVLDB Reference Format:
Yuxiang Zeng, Yongxin Tong, and Lei Chen. Last-Mile Delivery Made Practical: An Efficient Route Planning Framework with Theoretical Guarantees. *PVLDB*, 13(3): 320-333, 2019.
DOI: https://doi.org/10.14778/3368289.3368297

1. INTRODUCTION

Last-mile delivery (LMD) is defined as the movement of goods from a transportation origin to the final delivery destination [56]. In recent years, last-mile delivery services have been widespread in applications like urban logistics (*e.g.*, FedEx [5] and Cainiao [3]), e-commerce (*e.g.*, Amazon [2] and Alibaba [1]), food delivery (*e.g.*, Seamless [11] and Meituan [8]), etc. In these applications, *couriers* are often responsible to transport the goods (*e.g.*, parcels or food) for the *requesters* (*e.g.*, customers). The *origins* of the requests can be the warehouses in e-commerce or the restaurants in food delivery platforms, while the *destinations* can be the workplaces or homes of the requesters. Thus, a fundamental problem is *how to plan the routes (i.e., sequences of origins and destinations) among the couriers and requests.*

Such a route planning problem is not only strongly NP-hard [12] to address [22], but also has a large number of requests in real-world applications. Many solutions have been proposed to pursue approximate results for different objectives. Major studies [46, 37, 41, 34, 16, 29, 57, 39, 17, 27] take couriers into consideration by minimizing their travel time, *i.e.*, the makespan (maximum travel time) or the total travel time of the couriers. A few studies [20, 30, 21, 50] focus on alleviating the requesters' pain of waiting, *i.e.*, their total latency. Other studies [28] consider a more complex optimization goal, which is defined as the weighted sum of the couriers' objective and requesters' objective. However, these studies have the following limitations.

Limitation 1. Though they are effective in their concerned objectives, such effectiveness may stem from the great sacrifices of the efficiency or other important objectives. Currently, metaheuristics are usually used to achieve the effectiveness in both requesters' objective and couriers' objective (see Table 5 in the survey [28]). However, they usually consume longer running time in large-scale datasets. For example, the widely used ALNS algorithm [39, 25] can be 112×-168218× slower than [50, 46] in our experiments. Other efficient algorithms are only designed based on either couriers' objective or requesters' objective, which lose the effectiveness in terms of the other objective. For instance, [50] aims to minimize the total latency of requests and [46] focuses on minimizing the total travel time. However, in our experiments on real datasets, [50] has up to 82× longer travel time than [46] and [46] has up to 9.6× longer makespan and 8.6× higher total latency than the ALNS algorithm [25]. Therefore, it is still unknown *whether these methods can be efficient in large-scale data while simultaneously effective in the objectives of both requesters and couriers.*

Limitation 2. No existing solutions have theoretical guarantees for either objective in practical cases. Specifically, only [16, 21, 50] devise approximation solutions to plan the routes for *multiple* couriers. However, the establishment of theoretical guarantees usually relies on the restriction of the courier's capacity, *e.g.*, one [21], two [16], infinite capacity [50]. Thus, it is still an open problem *whether there*

Proceedings of the VLDB Endowment, Vol. 13, No. 3
ISSN 2150-8097.
DOI: https://doi.org/10.14778/3368289.3368297

exists a theoretically guaranteed solution to plan the routes for multiple couriers in practical cases.

To address *Limitation 1*, we study the Last-Mile Delivery (LMD) problem which focuses on route planning for *two* objectives, *i.e.*, minimizing the makespan (*i.e.*, maximum travel time) of couriers and the total latency of requesters. As labor laws of many countries [7, 14] restrict the maximum working time for the labors (*e.g.*, couriers), the last-mile delivery platform should also ensure the makespan of the couriers to be minimized with higher priority. By contrast, minimizing total travel time may cause some couriers to need longer travel time than others. However, as mentioned above, we also test whether these two objectives stem from a great sacrifice of total travel time in the experiments.

To solve *Limitation 2*, we propose a general framework which simultaneously has theoretical guarantees in two objectives. The framework applies an *adaptively increased travel budget* strategy to approximate the minimum makespan, which iteratively plans a segment of the final route under the travel budget for each courier. To approximate the minimum total latency, we need to deliver as many requests (denoted by k) as possible in each segment, which relies on the method to a subproblem called kLMD. As it has never been studied before, we also devise an effective method to kLMD and analyze the approximation ratio.

Our main contributions are summarized as follows:

- We propose a general framework and give a theoretical analysis in terms of both makespan and total latency. To the best of our knowledge, we are the first to use **one** approach to **simultaneously** guarantee **two** objectives in the Last-Mile Delivery (LMD) problem.
- We devise an effective algorithm ESI for the subproblem kLMD, which is critical to the effectiveness of the framework. As the approximation ratio of ESI is $O(\log n)$, we complete our approach with both approximation ratios of $O(6 \log n) \sim O(\log n)$, where n is the number of requests. The logarithmic approximation ratios are by far the best guarantees [17, 27, 28].
- Extensive experiments show the superior effectiveness of our approach. It outperforms the state-of-the-art methods [46, 25, 17, 50] in terms of both makespan and total latency with averagely 48.4%-96.0% and 49.7%-96.1% improvements. Especially in large-scale real datasets, our algorithm yields 29.3×-108.9× shorter makespan and 20.2×-175.1× times lower total latency than the state-of-the-art algorithms.

In the following, we present the definition of the LMD problem in Sec. 2 and review related work in Sec. 3. Then we introduce our framework for the problem in Sec. 4 and our solution to its subproblem in Sec. 5. Finally, we conduct experiments in Sec. 6 and conclude in Sec. 7.

2. PROBLEM STATEMENT

2.1 Preliminaries

Definition 1 (Request). *A request is denoted by $r = \langle o_r, d_r, c_r \rangle$, which is initially located at the origin o_r and needs to be delivered at the destination d_r. The weight of the goods in this request is c_r.*

We use R to denote a set of n requests. To **complete** a request, a courier needs to first pick up the goods at the origin and then deliver it at the destination. We use e_r to denote the **latency** of the request, *i.e.*, the time when the requester receives the goods from the courier at the destination.

Definition 2 (Courier). *A courier is denoted by $w = \langle o_w, c_w \rangle$, who is initially located at o_w with a capacity c_w.*

We use W to denote a set of m couriers and R_w to denote a set of requests assigned to the courier w by the platform.

Definition 3 (Route). *Given a courier w and the assigned requests R_w, a route of this courier is denoted by $S_w = \langle l_0, l_1, l_2, \cdots, l_N \rangle$, which is a sequence of the initial location of courier w and all the origins and the destinations of the requests in R_w, i.e., $l_0 = o_w$ and $l_i \in \{o_r | r \in R_w\} \cup \{d_r | r \in R_w\}$ for all $1 \le i \le N$.*

Accordingly, the courier needs to start from his/her initial location and then go through additional n places. The route is **feasible** if the following constraints are satisfied.

(1) **Order Constraint.** For every request $r \in R_w$, o_r lies ahead of d_r in the route, *i.e.*, a request needs to be picked up before delivered.
(2) **Capacity Constraint.** At any time, the total weight of all the goods that have been picked up but not delivered does not exceed the capacity of courier w.
(3) **Completion Constraint.** At the end of the route, all the requests need to be delivered (*i.e.*, completed).

Similar to the existing works [17, 27, 46, 43, 31], we focus on the metric space (V, d) where V contains the locations of both requests and couriers, and $\mathsf{d} : V \times V \to [0, +\infty)$ is the function of the travel time between any two locations. For example, a metric space can be a Euclidean space or a road network. Thus, we use $D(S_w)$ to denote the travel time of the route S_w, *i.e.*, $D(S_w) = \sum_{i=1}^{N} \mathsf{d}(l_{i-1}, l_i)$.

2.2 Problem Definition

Based on the basic concepts above, we first define the *Last-Mile Delivery (LMD)* problem as follows.

Definition 4 (LMD Problem). *Given a set of couriers W and a set of requests R, we aim to plan a route S_w for each courier $w \in W$ while simultaneously minimizing the following objectives:*

*(1) **makespan** (a.k.a, maximum travel time) of the couriers, $\mathrm{OBJ}_1(W, R) = \max_{w \in W} D(S_w)$, where $D(S_w)$ is the travel time of the route S_w;*
*(2) **total latency** of the requests, $\mathrm{OBJ}_2(W, R) = \sum_{r \in R} e_r$, where e_r is the latency of the request r.*

and meet the following constraints:

- ***Feasibility constraint.** Each route S_w is feasible.*
- ***Completion constraint.** All the requests in R should be completed, i.e., $\bigcup_{w \in W} R_w = R$.*

Next, we illustrate the LMD problem in Example 1.

Example 1. *Suppose there are 2 couriers w_1, w_2 and 5 requests r_1-r_5 on a last-mile delivery platform. As shown in Fig. 1, the couriers are initially located at $(0, 0)$ and $(1, 0)$ of a Euclidean space whose capacities are 3 and speeds are 1. The origins, destinations and weights of the requests are listed in Table 1. In the LMD problem, we suppose the platform assigns r_1-r_3 to w_1 and r_4, r_5 to w_2, i.e., $R_{w_1} = \{r_1, r_2, r_3\}$ and $R_{w_2} = \{r_4, r_5\}$. Fig. 1 shows a plan of the routes with the travel time of each edge annotated beside it. The two routes are $S_{w_1} = \langle o_{w_1}, o_{r_1}, d_{r_1}, o_{r_2}, d_{r_2}, o_{r_3}, d_{r_3} \rangle$ and $S_{w_2} = \langle o_{w_2}, o_{r_4}, o_{r_5}, d_{r_5}, d_{r_4} \rangle$. Therefore, $D(S_{w_1}) = 1.4+3.6+2.8+3.6+2+2 = 15.4$, $D(S_{w_2}) = 3.2+1.4+1 = 5.6$,*

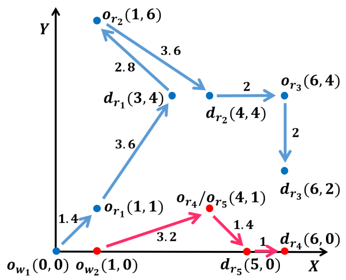


Figure 1: An illustration of the LMD problem.

Table 1: The set of requests with weight $c_{r_i} = 1$.

Request	r_1	r_2	r_3	r_4	r_5
origin o_r	(1,1)	(1,6)	(6,4)	(4,1)	(4,1)
destination d_r	(3,4)	(4,4)	(6,2)	(6,0)	(5,0)

and then $\text{OBJ}_1(W,R) = \max\{D(S_{w_1}), D(S_{w_2})\} = 15.4$. *In addition, we can calculate the latency of the requests as* $e_{r_1} = d(o_{w_1}, o_{r_1}) + d(o_{r_1}, o_{d_1}) = 1.4 + 3.6 = 5$, $e_{r_2} = 11.4$, $e_{r_3} = 15.4$, $e_{r_4} = 5.6$ *and* $e_{r_5} = 4.6$. *Thus,* $\text{OBJ}_2(W,R) = e_{r_1} + \cdots + e_{r_5} = 42$. *The LMD problem aims to find the optimal routes which minimize the makespan and the total latency at the same time. In this example, the optimal routes are* $S_{w_1} = \{o_{w_1}, o_{r_1}, d_{r_1}, o_{r_2}, d_{r_2}\}$ *and* $S_{w_2} = \{o_{w_2}, o_{r_4}, o_{r_5}, d_{r_4}, d_{r_5}, o_{r_3}, d_{r_3}\}$*, which simultaneously obtain both minimum makespan and minimum total latency.*

Hardness Results. When minimizing either of the objectives, the LMD problem is equivalent to a variant of the classic dial-a-ride problem [55]. Thus, our LMD problem is both **NP-hard** and **APX-hard** due to the hardness of the dial-a-ride problem. We refer readers to [22] for the proof of the NP-hardness. Table 2 lists the major notations.

3. RELATED WORK

Our LMD problem originates from the dial-a-ride problem, which is widely applied in ridesharing and last-mile delivery. The dial-a-ride problem focuses on planning the routes for a set of origins and destinations and has recently attracted extensive research interests from the database [38, 29, 37, 46, 57, 41, 19], transportation science [28, 25, 39] and other communities [16, 21, 50, 27]. We categorize existing solutions into *heuristic* and *approximation* algorithms.

(1) Heuristic Algorithms. Many metaheuristics have been proposed to solve this problem, *e.g.*, simulated annealing, tabu search, genetic algorithm (please refer to survey [28] for details). These studies usually use a weighted sum of the couriers' objective and requesters' objective as the optimization goal in their problems [28]. Among these methods, adaptive large neighborhood search [39, 25] is one of the most widely used solutions (*e.g.*, [39] has over 1000 citations in Google Scholar) and has also been applied in the industry (*e.g.*, Cainiao [3, 13]). In a reasonable running time, the method can achieve good effectiveness when the number of requests is median-scale. However, the required time cost will dramatically increase in last-mile delivery (see Sec. 6), where there are usually a large number of requests.

Recently, many methods from the **database community** are proposed to plan the routes in large-scale spatial data [37, 38, 29, 41, 19, 46, 57]. These solutions are designed based on the efficient *insertion* operation to minimize the travel time of couriers, *e.g.*, T-Share [37, 38], kinetic [29, 41, 19] and pruneGreedyDP [46]. In these studies, the *insertion* operation selects the optimal positions to insert the origin

Table 2: Summary of major notations.

Notation	Description
r, R	a request and a set of requests
o_r, d_r	the origin and destination of r
c_r, e_r	the weight and latency of r
w, W	a courier and a set of couriers
o_w, c_w	initial location and capacity of w
R_w	the set of requests assigned to w
S_w	the route of courier w
OBJ_1	minimize makespan of the couriers
OBJ_2	minimize total latency of the requests
n	the number of requesters (*i.e.*, $\|R\|$)
m	the number of couriers (*i.e.*, $\|W\|$)

and destination of a request into the current route of one courier such that the increased travel time of the new route is minimized. Tong *et al.* [46] and Xu *et al.* [57] design different insertion algorithms with linear time complexity, which is currently the fastest. Though these studies are efficient to handle large-scale requests, the planned routes may have longer total latency, which will result in bad experiences for the requesters in last-mile delivery.

(2) Approximation Algorithms. There are relatively fewer studies on approximation algorithms with theoretical guarantees. Specifically, [16] aims to minimize the total travel time of the couriers, [17, 27] aim to minimize the makespan of the couriers, and [21, 50] aim to minimize the total latency of the requesters. However, the approximation ratios in [16, 21, 50] rely on the special cases of capacity (*i.e.*, c_w), *e.g.*, one [21], two [16] or infinite capacity [50]. Though other studies [17, 27] do not have such assumptions, they focus on the special case of a single courier (*i.e.*, $m = 1$). Among these studies, [17] has the best approximation ratio.

Last-mile delivery (*e.g.*, urban logistics and food delivery) is also viewed as one of the killer applications in Spatial Crowdsourcing (SC) (please refer to survey [48, 26] or tutorial [42] for details). Related studies in spatial crowdsourcing focus on different problems in last-mile delivery. Specifically, [58, 20, 43, 59] aim to assign one proper request (*i.e.*, a task in SC) for each courier (*i.e.*, a worker in SC) instead of planning the routes. [36, 18] focus on selecting the suitable taxi drivers to ship the goods in last-mile delivery. [33] studies a different food delivery scenario, where the couriers are allowed to select the requests by themselves. [30] focuses on clustering the food delivery orders such that the total latency can be minimized. Therefore, their methods cannot be used in our LMD problem.

4. FRAMEWORK FOR LMD PROBLEM

In this section, we introduce our general framework for the LMD problem. Specifically, we explain our basic idea in Sec. 4.1, describe the algorithm details in Sec. 4.2 and present our approximation analysis in Sec. 4.3.

4.1 Basic Idea

Our LMD problem has two objectives, *i.e.*, minimizing both makespan (*i.e.*, maximum travel time) of couriers and total latency of requests. In order to achieve that, our basic idea is based on an **adaptively increased travel budget**. *(1)* To minimize the **makespan**, we need to balance the travel time of each route. Specifically, we set the same **travel budget** for all couriers and then iteratively plan a segment of the final route for each courier. The iterations terminate when all the requests are assigned.

(2) To minimize the **total latency**, we first need to deliver **as many requests as possible** in each segment.
(3) To determine the value of the travel budget, we use an **adaptively increased** travel budget instead of a static one. Accordingly, the courier can complete the requests that are nearer to him when the budget is smaller. After that, he/she can cooperate with others by delivering the faraway requests when the budget gets larger.

To deliver as many requests as possible under a travel budget, we first enumerate the value of k from 1 to the total number n and then plan a route to deliver exactly k requests while minimizing the travel time. If its travel time is under the budget, we can try a larger k. Thus, we define a subproblem called k Last-Mile Delivery (kLMD) problem.

Definition 5 (kLMD Problem). *Given a courier w, a set of requests R and an integer k, we aim to plan a route S_w^k for completing exactly k requests R_w^k to minimize the* ***total travel time*** *$D(S_w^k)$ and meet the following constraints:*

- ***Feasibility constraint.*** *The route S_w^k is feasible.*
- ***k-Completion constraint.*** *Exactly k requests in R are completed, i.e., $R_w^k \subseteq R$ and $|R_w^k| = k$.*

The kLMD problem is NP-hard since it generalizes an NP-hard problem, the dial-a-ride problem [22], by $k = n$. We will elaborate our approximation solution to the subproblem in Sec. 5 and temporarily assume that a ρ-approximation algorithm for the kLMD problem exists in the following.

4.2 Framework Details

Algorithm Details. In lines 1-2, we use R' to denote the set of currently unassigned requests. In line 3, we initialize the travel budget δ as ρ and iteratively double it. For the given travel budget, we find the maximum integer k between 1 and $|R'|$ for each courier w, such that it takes no more than δ time to complete k requests (denoted by R_w^k) from the unassigned ones in the route S_w^k (lines 4-5). The maximum integer k is denoted by k^*. If such k^* exists and some requests in $R_w^{k^*}$ have not been assigned yet (line 6), we update the route S_w of w to complete such requests. In line 7, we first refine the route $S_w^{k^*}$ by removing the initial location of the courier since he/she can directly go to the origin of the first request in R_w^k. In line 8, we update the route S_w by first following the original route S_w and then following the newly planned route S_w^{k*}, *i.e.*, we append S_w^{k*} at the end of S_w. Finally, if all the requests have already been assigned, we return the final route for each courier (lines 10-11).

Example 2. *Back to Example 1. For simplicity, we assume $\rho = 1$. When $\delta = 1, 2, 4$, neither w_1 nor w_2 can complete any request. Thus, no request is assigned. When $\delta = 8$, $k^* = 1$, $R_{w_1}^{k^*} = \{r_1\}$, $S_{w_1}^{k^*} = \langle o_{w_1}, o_{r_1}, d_{r_1}\rangle$, $D(S_{w_1}^{k^*}) = 1.4 + 3.6 = 5 \leq 8$ (line 5). Thus, $S_{w_1} = \langle o_{w_1}, o_{r_1}, d_{r_1}\rangle$ (line 8). Similarly, since $S_{w_2}^{k^*} = \langle o_{w_2}, o_{r_4}, o_{r_5}, d_{r_5}, d_{r_4}\rangle$, $D(S_{w_2}^{k^*}) = 3.2+1.4+1 = 5.6 \leq 8$ (line 5), $S_{w_2} = \langle o_{w_2}, o_{r_4}, o_{r_5}, d_{r_5}, d_{r_4}\rangle$ (line 8). At this time, the set of unassigned requests is $R' = \{r_2, r_3\}$ (line 9). When $\delta = 16$, $S_{w_1}^{k^*} = \langle o_{w_1}, o_{r_2}, d_{r_2}, o_{r_3}, d_{r_3}\rangle$ $D(S_{w_1}^{k^*}) = 13.7 \leq 16$ (line 5). After line 8, we have $S_{w_1} = \langle o_{w_1}, o_{r_1}, d_{r_1}, o_{r_2}, d_{r_2}, o_{r_3}, d_{r_3}\rangle$. Finally, all of the requests have been assigned (lines 10-11).*

Complexity Analysis. As a courier can not complete all the n requests in practice, we assume a courier finally takes at most $\mathcal{R}$ ($\ll n$) requests. Suppose the time complexity and space complexity of a ρ-approximation algorithm for kLMD problem is T and S respectively. Thus, there are

Algorithm 1: General Framework

```
   input : requests $R$, couriers $W$, and a parameter $\rho$
   output: a route $S_w$ for each courier $w$
1  currently unassigned requests $R' \leftarrow R$;
2  currently planned route $S_w \leftarrow \emptyset$ for each courier $w$;
3  foreach travel budget $\delta \in \{\rho \cdot 2^0, \rho \cdot 2^1, \rho \cdot 2^2, \cdots\}$ do
4      foreach courier $w \in W$ do
5          $k^* \leftarrow$ maximum integer $k \in [1, |R'|]$ such that
             $(S_w^k, R_w^k) \leftarrow k\mathtt{LMD}(w, R', k)$ and $D(S_w^k) \leq \delta$;
6          if $k^*$ exists and $R_w^{k^*} \neq \emptyset$ then
7              if $S_w \neq \emptyset$ then remove $o_w$ from $S_w^{k^*}$;
8              $S_w \leftarrow$ append $S_w^{k^*}$ at the end of $S_w$;
9              $R' \leftarrow R' - R_w^{k^*}$;
10     if requests $R$ are all assigned then break;
11 return $\{S_w | w \in W\}$;
```

$O(\log n)$ iterations in lines 3-10 since the shortest travel time to complete all the requests is polynomial-time of n. In lines 4-9, there are $O(m)$ iterations. In each iteration, line 5 takes $O(\mathcal{R} \cdot T)$ time and lines 6-9 take $O(\mathcal{R})$ time. Finally, the time complexity of Algo. 1 is $O(\mathcal{R} m \log n \cdot T)$ and its space complexity is $O(S + n + m)$.

4.3 Approximation Analysis

In the following, we first define the (α, β)-approximation algorithm for the LMD problem as follows. Intuitively, we compare each objective of the approximation algorithm with the result of each optimal solution that only optimizes the corresponding objective.

Definition 6 ((α, β)-Approximation). *Let OBJ_i denote the value of the i-th objective given by the approximation algorithm and OBJ_i^* be the value of the i-th objective given by the optimal method for this objective. An algorithm for the LMD problem is an (α, β)-approximation method such that for any instance (W, R), we have (1) $\mathrm{OBJ}_1(W, R) \leq \alpha \cdot \mathrm{OBJ}_1^*(W, R)$ and (2) $\mathrm{OBJ}_2(W, R) \leq \beta \cdot \mathrm{OBJ}_2^*(W, R)$.*

Based on the above definition, we present the approximation ratios of the framework in Theorem 1.

Theorem 1. *Given a **ρ**-approximation algorithm for the kLMD problem, our framework Algo. 1 achieves an approximation ratio of ($\boldsymbol{6\rho, 6\rho}$).*

The **main idea** to prove Theorem 1 is as follows:
(1) We introduce Lemma 1 to show that the total travel time of our planned route S_w for a courier w is bounded by the shortest route to complete the same set of requests R_w;
(2) Accordingly, we can prove the α and β of the two objectives in Lemma 2 and Lemma 3 respectively;
(3) We directly prove Theorem 1 based on the lemmas.

Accordingly, we first present Lemma 1 in the following.

Lemma 1. *Given a ρ-approximation method for the kLMD problem, a courier w with his/her assigned requests R_w, let S_w be the route obtained by Algo. 1 and S_w^* be the optimal route with the shortest travel time to complete the requests R_w by the courier w. We have $D(S_w) \leq 6\rho \cdot D(S_w^*)$.*

Proof. Without loss of generality (WLOG), we consider a courier w and assume that $D(S_w^*) \in [2^{\mathcal{L}-1}, 2^{\mathcal{L}})$ for some integer $\mathcal{L} \geq 1$ (*w.r.t.* the courier w). Based on the definition of the kLMD problem, all the requests in R_w can be completed by w in Algo. 1 when the travel budget is $\rho \cdot 2^{\mathcal{L}}$. Since we always update the route S_w by appending the newly planned trip (line 8), the travel time of S_w is monotonically increasing when the travel budget increases from $\rho \cdot 2^0$ to $\rho \cdot 2^{\mathcal{L}}$. For example, we assume the current travel budget is $\rho \cdot 2^i$, *i.e.*, $D(S_w^{k^*}) \leq \rho \cdot 2^i$. In line 7, the travel time of the trip connecting the last location of S_w and the first location of $S_w^{k^*}$ is no longer than the travel time of the trip, where the courier first returns to his/her initial location at the end of S_w and then follows the newly appended route $S_w^{k^*}$ (due to triangle inequality). Since the travel time of return trip is under the previous travel budget $\rho \cdot 2^{i-1}$, we have

$$D(S_w) \leq \left[\sum_{i=0}^{\mathcal{L}-1} (2 \cdot \rho \cdot 2^i) \right] + \rho \cdot 2^{\mathcal{L}} < 3\rho \cdot 2^{\mathcal{L}} \leq 6\rho \cdot D(S_w^*). \quad (1)$$

□

In Lemma 2 and Lemma 3, we elaborate on our **approximation analysis** of Algo. 1 in terms of α and β.

LEMMA 2. *Given a ρ-approximation method for the kLMD problem, our framework Algo. 1 achieves an approximation ratio of 6ρ to minimize the* ***makespan****.*

PROOF. Let S_w^* be the route of courier w in the optimal solution to minimizing the **makespan** and R_w^* be the corresponding requests assigned to the courier w according to the route. WLOG, we assume that the w' is the courier with the maximum travel time in the optimal solution, *i.e.*, $w' = \arg\max_w D(S_w^*)$. Similar to Lemma 1, we assume some integer $\mathcal{L}$ such that $D(S_{w'}^*) \in [2^{\mathcal{L}-1}, 2^{\mathcal{L}})$. In the following, we want to prove that all requests must be assigned by Algo. 1 when the travel budget δ increases to $\rho \cdot 2^{\mathcal{L}}$.

Since the courier w' takes the maximum travel time, we know that the travel time of any S_w^* is also bounded by $2^{\mathcal{L}}$, *i.e.*, $D(S_w^*) \leq D(S_{w'}^*) < 2^{\mathcal{L}}$. Then for any courier w, his/her assigned requests R_w^* in the optimal solution must be delivered in our planned route S_w when the travel budget δ becomes $\rho \cdot 2^{\lceil D(S_w^*) \rceil} \leq \rho \cdot 2^{\mathcal{L}}$. Some of the requests in R_w^* may also be assigned to others in the earlier iterations. Therefore, all the requests must be assigned by Algo. 1 when the travel budget becomes $2^{\mathcal{L}}$.

As we have $D(S_w) \leq 3\rho \cdot 2^{\mathcal{L}}$ for each courier w according to Eq. (1) in Lemma 1, we derive the following upper bound and lower bound:

$$\text{OBJ}_1(W,R) = \max_{w \in W} D(S_w) \leq 6\rho \cdot 2^{\mathcal{L}-1}$$
$$\text{OBJ}_1^*(W,R) = \max_{w \in W} D(S_w^*) = D(S_{w'}^*) \geq 2^{\mathcal{L}-1}$$

Finally, we have $\text{OBJ}_1(W,R) \leq 6\rho \cdot \text{OBJ}_1^*(W,R)$ for any instance (W,R), *i.e.*, $\alpha \leq 6\rho$. □

LEMMA 3. *Given a ρ-approximation method for the kLMD problem, our framework Algo. 1 achieves an approximation ratio of 6ρ to minimize the* ***total latency****.*

PROOF. Let S_w^* be the route of courier w in the optimal solution to minimizing the **total latency** and R_w^* be the corresponding requests assigned to the courier w according to the route. WLOG, we use function $N^*(R, 2^i)$ to denote the number of requests in R whose latency $e_r^* \in [2^{i-1}, 2^i)$ by the optimal solution. Similarly, we use function $N(R, 3\rho \cdot 2^i)$ to denote the number of requests in R whose latency $e_r \in [3\rho \cdot 2^{i-1}, 3\rho \cdot 2^i)$ by Algo. 1. In the following, we first prove a powerful result. For any integer $j \geq 0$, the number of requests whose latency is strictly shorter than 2^j by the optimal solution is no more than the number of requests whose latency is strictly shorter than $3\rho \cdot 2^j$ by Algo. 1, *i.e.*,

$$\forall j \in \mathbb{Z}, \sum\nolimits_{i=0}^{j} N^*(R, 2^i) \leq \sum\nolimits_{i=0}^{j} N(R, 3\rho \cdot 2^i). \quad (2)$$

We use n_w^* to denote the number of requests assigned to the courier w whose latency is shorter than 2^j in the optimal solution. Thus, the shortest travel time to complete these requests is at most 2^j. Otherwise, the last delivered request must have a longer latency than 2^j. Since a ρ-approximation algorithm for the kLMD problem is given, it can achieve a feasible route (with parameter n_w^*) in line 5 such that the travel time of this route is bounded by $\rho \cdot 2^j$. In other words, at least n_w^* requests must have been assigned when the budget δ increases to $\rho \cdot 2^j$. Thus, we derive the lower bound of the number of currently assigned requests R' as

$$\text{when } \delta = \rho \cdot 2^j, |R'| \geq \sum\nolimits_{w \in W} n_w^* = \sum\nolimits_{i=0}^{j} N^*(R, 2^i). \quad (3)$$

Next, we need to prove the latency of any request $r \in R'$ is strictly shorter than $3\rho \cdot 2^j$. According to Eq. (1), we know the total travel time of any courier is currently bounded by $3\rho \cdot 2^j$. Hence, the maximum latency of the requests is shorter than $3\rho \cdot 2^j$, *i.e.*, $|R'| = \sum_{i=0}^{j} N(R, 3\rho \cdot 2^i)$.

Similar to Lemma 2, we assume the last budget is $\rho \cdot 2^{\mathcal{L}}$. Then we derive the upper bound and lower bound as:

$$\text{OBJ}_2(W,R) \leq \sum\nolimits_{i=0}^{\mathcal{L}} \big(N(R, 3\rho \cdot 2^i) \cdot (3\rho \cdot 2^i)\big),$$
$$\text{OBJ}_2^*(W,R) \geq \sum\nolimits_{i=0}^{\mathcal{L}} \big(N^*(R, 2^i) \cdot 2^{i-1}\big).$$

Thus, we can calculate the ratio β as

$$\beta = \frac{\text{OBJ}_2(W,R)}{\text{OBJ}_2^*(W,R)} \leq \frac{6\rho \cdot \sum_{i=0}^{\mathcal{L}} \big(N(R, 3\rho \cdot 2^i) \cdot 2^{i-1}\big)}{\sum_{i=0}^{\mathcal{L}} \big(N^*(R, 2^i) \cdot 2^{i-1}\big)}. \quad (4)$$

As we know that $\sum_{i=0}^{\mathcal{L}} N(R, 3\rho \cdot 2^i) = \sum_{i=0}^{\mathcal{L}} N^*(R, 2^i) = n$ (*i.e.*, all the requests are completed eventually) and prove that Eq. (2) holds for $j = 0, \cdots, \mathcal{L}$, we have

$$\sum\nolimits_{i=0}^{\mathcal{L}} \big(N(R, 3\rho \cdot 2^i) \cdot 2^{i-1}\big) \leq \sum\nolimits_{i=0}^{\mathcal{L}} \big(N^*(R, 2^i) \cdot 2^{i-1}\big).$$

Finally, we complete our proof as $\beta \leq 6\rho$. □

5. ALGORITHM FOR *K*LMD PROBLEM

In this section, we introduce our solution to the subproblem kLMD in our framework. Specifically, we introduce the preprocessing procedure in Sec. 5.1, elaborate on our basic idea in Sec. 5.2 and explain algorithm details in Sec. 5.3. Finally, we analyze the approximation ratio in Sec. 5.4 and discuss some practical issues in Sec. 5.5.

5.1 Preprocessing

In Sec. 5.1.1, we introduce the spatial index called hierarchically separated tree (HST) [15, 23], which can embed any metric into a balanced tree HST. In Sec. 5.1.2, we then transform the shortest path between origin and destination of each request on the original metric into a path on HST.

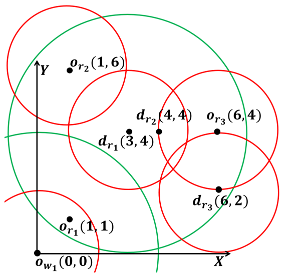
5.1.1 Spatial Index

Construction. The index HST is usually used to project a set of objects in databases [43, 20, 32, 24]. The **main idea** of constructing an HST is to decompose the objects into clusters according to the original distances.

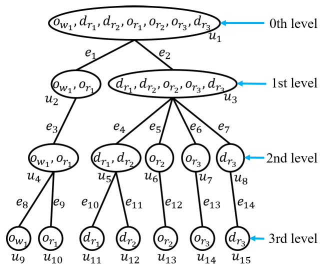
The **algorithm details** of the construction procedure [23] is illustrated in Algo. 2. In line 1, it randomly generates a permutation order π of the location set V and a parameter γ. In line 2, Δ denotes the longest travel time between any two locations and $\mathcal{H}$ denotes the height of the HST. At the i-th level of HST, we use a *cluster* to denote the set of locations that are in the same circular range with a radius $\mathcal{L}_i$ (*i.e.*, $\mathsf{d}(\pi(j), u) \leq \mathcal{L}_i$ in line 7) , and C_i denotes the set of such clusters. Since each cluster corresponds to a node on

Algorithm 2: Spatial Index Construction

```
  input  : a metric space $(V, \mathsf{d})$
  output: an HST $T$
1 sample a permutation $\pi$ of $V$ and $\gamma \in [0.5, 1]$;
2 $\Delta \leftarrow \max_{u,v} \mathsf{d}(u, v), \mathcal{H} \leftarrow \lceil \log_2 \Delta \rceil, C_0 \leftarrow V$;
3 for $i \leftarrow 1$ to $\mathcal{H}$ do
4     $\mathcal{L}_i \leftarrow \gamma \cdot 2^{\mathcal{H}-i}, C_i \leftarrow \emptyset$;
5     foreach cluster $c \in C_{i-1}$ do
6         for $j \leftarrow 1$ to $|V|$ do
7             cluster $c' \leftarrow \{u \in c | \mathsf{d}(\pi(j), u) < \mathcal{L}_i\}$;
8             add $c'$ into $C_i$ and remove $u \in c'$ from $c$;
9 return $T \leftarrow$ create an HST with $C_0, \cdots, C_{\mathcal{H}}$;
```



(a) The decomposition.



(b) The corresponding HST.

Figure 2: An illustration of constructing the HST.

the HST, C_i also denotes the set of nodes on the i-th level. In lines 3-8, it performs a hierarchical decomposition to get C_i based on the specific order $\pi(j)$ in line 6. At the i-th level, we decompose the clusters $c \in C_{i-1}$ into new clusters c' by smaller circular ranges with shorter radii in lines 7-8. In line 9, we create an HST with all the cluster sets.

Example 3. *Suppose we construct the HST with locations of* w_1 *and* r_1-r_3. *Fig. 2 illustrates the construction procedure. We also assume* $\gamma = 1$ *and* $\pi = \langle o_{w_1}, d_{r_1}, d_{r_2}, o_{r_1}, o_{r_2}, o_{r_3}, d_{r_3} \rangle$. *In line 2, we have* $\Delta = \mathsf{d}(o_w, o_{r_3}) = 7.2, \mathcal{H} = \lceil \log_2 \Delta \rceil = 3$. *As shown in Fig. 2b,* C_0 *only has one node,* i.e., *the root of the HST which contains all locations. In line 3, when* $i = 1$, $\mathcal{L}_i = 1 \cdot 2^{3-1} = 4$. *The circles marked by green in Fig. 2a represents the range of* $\mathsf{d}(o_{w_1}, u) < 4$ *and* $\mathsf{d}(d_{r_1}, u) < 4$ *respectively (line 7). Since* $\mathsf{d}(o_w, o_w) = 0 < 4$ *and* $\mathsf{d}(o_{r_1}, o_w) = 1.41 < 4$, *the cluster* c' *only contains* o_{w_1} *and* o_{r_1} *in line 7. In the end,* C_1 *has two nodes at the first level of HST, as shown in Fig. 2b. Similarly, we further decompose the locations using smaller circular ranges with radii of 2 which are marked by red. At last,* C_2 *has five nodes and* C_3 *has seven nodes on the HST in Fig. 2b.*

Complexity. In Algo. 2, the number of iterations in line 3, line 5 and line 6 are $\mathcal{H}$, $|V|$ and $|V|$, respectively. Thus, the time complexity is $O(\mathcal{H}|V|^2)$ and the space complexity is $O(\mathcal{H}|V|)$, where $\mathcal{H}$ denotes the height of the HST and $|V|$ denotes the number of locations.

Properties. Based on Algo. 2, the HST is a balanced tree and hence inherits all the sound properties of a balanced tree. Besides, it also has the following powerful properties: *(1)* It guarantees that $\mathsf{d}_T(u, v) \geq \mathsf{d}(u, v)$ and $\mathbb{E}[\mathsf{d}_T(u, v)] \leq O(\log |V|) \cdot \mathsf{d}(u, v)$, where u, v are two locations in the location set V, $\mathsf{d}(u, v)$ is the travel time between u and v in the original metric, and $\mathsf{d}_T(u, v)$ is the travel time on the HST. *(2)* Each leaf node is a cluster containing a unique location in V, while each non-leaf node contains a subset of V, which is the union set of locations contained in its children. *(3)* For any nodes u, v at the i-th level (u can be same as v), u and v have the same travel time (*i.e.*, $\mathcal{L}_i$) to their children $\mathsf{ch}(u)$ and $\mathsf{ch}(v)$. *i.e.*, $\mathsf{d}_T(u, \mathsf{ch}(u)) = \mathsf{d}_T(v, \mathsf{ch}(v))$.
(4) For any node v, let $\mathsf{pa}(v)$ and $\mathsf{ch}(v)$ be its parent and its child, we have $\mathsf{d}_T(v, \mathsf{pa}(v)) = 2 \cdot \mathsf{d}_T(v, \mathsf{ch}(v))$.

5.1.2 Shortest Path Transformation

To minimize the total travel time, an effective algorithm tries to makes the requests share the routes of the requests between their origins and destinations. However, it is quite hard to determine which subsets of requests are suitable to share together on the original metric since there may be many shortest paths or approximate shortest paths between the origins and destinations in practice. By our index, we can transform such paths into a unique path on the HST.

Basic Idea. According to the first two properties mentioned in Sec. 5.1.1, we know: *(1)* the travel time on the HST is always no shorter than the travel time on the original metric, and *(2)* each leaf node of the HST corresponds to a unique location. Thus, the shortest path between an origin and a destination is actually a path (denoted by $\mathsf{path}(.,.)$) between their corresponding leaf nodes on the HST.

Algorithm Sketch. Since each leaf node corresponds to a unique location, we can easily obtain the path $\mathsf{path}(.,.)$ by traversing the HST from the two leaf nodes up to their least common ancestor (denoted by $\mathsf{lca}(.,.)$). By iterating such process, we can transform the paths of multiple requests on the original metric into the paths on the HST.

Example 4. *In Fig. 2b, leaf nodes* u_{10}, u_{11} *and* u_{12} *correspond to* o_{r_1}, d_{r_1} *and* d_{r_2} *respectively. Thus,* $\mathsf{lca}(u_{10}, u_{11}) = \mathsf{lca}(u_{10}, u_{12}) = u_1$ *and* $\mathsf{path}(u_{10}, u_{11}) = \langle e_9, e_3, e_1, e_2, e_4, e_{10} \rangle$ $\mathsf{path}(u_{10}, u_{12}) = \langle e_9, e_3, e_1, e_2, e_4, e_{11} \rangle$. *Since* $\mathsf{path}(u_{10}, u_{11})$ *and* $\mathsf{path}(u_{10}, u_{12})$ *have many common edges, it might be better to deliver the corresponding requests together.*

Complexity Analysis. Both time complexity and space complexity is $O(\mathcal{H}n)$, where $\mathcal{H}$ is the height of HST.

5.2 Basic Idea

The **basic idea** of our algorithm ESI is as follows:

(1) We aim to select k requests that are closer to the initial location o_w and hence we pick them from the lowest subtree of HST that contains o_w and at least k requests. The reason is $\mathsf{d}_T(o_w, u) \geq 2 \cdot \mathsf{d}_T(o_w, v)$, where u is any location outside the subtree and v is any location inside the subtree.

(2) To plan the route, we apply the *insertion* operator [57], which has been widely used in recent works [46, 57, 29, 37] due to its superior performance. Insertion refers to the procedure of adding a new request into the current route, *i.e.*, inserting the origin and destination of the new request into the right positions. The final route is usually formed by sequentially inserting the requests. Thus, the *insertion order* is important, which is not discussed in these studies.

(3) The requests, which are more likely to share routes together, should be inserted consecutively. Intuitively, if two paths with the same $\mathsf{lca}(.,.)$ have no common edges on some level of the HST, they can not have any common edges on the lower levels. Thus, we can vectorize each path in a top-down manner. We next sort the requests based on the lexicographic order of the vectors. After that, the requests, which have common prefixes of their vectors (*i.e.*, common edges of the paths), are getting closer.

5.3 Algorithm ESI

Our algorithm ESI (Algo. 3) for the kLMD problem has three major steps: ***E****mbedding*, ***S****orting*, and ***I****nsertion*.

Algorithm 3: Algorithm ESI for kLMD problem

```
input  : a courier $w$, requests $R$ and parameter $k$
output: a route for the courier $S_w$
1 $S_w \leftarrow \emptyset, V \leftarrow \{o_w\} \cup \{o_r | r \in R\} \cup \{d_r | r \in R\}$;
2 $T \leftarrow$ embed metric $(V, \mathsf{d})$ into HST by Algo. 2;
3 $T' \leftarrow$ the lowest subtree of $T$ such that its root node
    contains $o_w$ and at least $k$ requests.;
4 $R^k \leftarrow$ the first $k$ requests during the depth-first
    search of $T'$ starting from the leaf node of $o_w$;
5 transform the paths of $r \in R^k$ into $\mathsf{path}(o_r, d_r)$;
6 label the edges in $\mathsf{path}(.,.)$ by integers based on the
    visited order during the previous search process;
7 encode each $\mathsf{path}(o_r, d_r)$ into a vector $\mathsf{vec}(o_r, d_r)$ with
    the labeled integers in a top-down manner;
8 $\widehat{R^k} \leftarrow$ sort the requests $R^k$ based on lexicographic
    order of $\mathsf{vec}(o_r, d_r)$ by bucket sort;
9 return $S_w \leftarrow$ sequentially insert $r \in \widehat{R^k}$ into $S_w$;
```

Algorithm Details. The detailed procedure of ESI is:
(1) **Metric Embedding.** In lines 1-2, we embed the original metric into the HST metric by Algo. 2.
(2) **Sorting.** In lines 3-4, we select the k requests from R. Specifically, we first find the lowest subtree T' of the HST such that the root node of subtree contains o_w and at least k requests (line 3). Accordingly, T' is the smallest subtree to cover the selected requests. Therefore, we select the first k requests R^k during the depth-first search of T' starting from the leaf node of o_w (line 4). Then we get the unique paths of the requests on HST in line 5, and label their edges with continuous integers based on the visiting order during the depth-first search in line 6. We next encode each path $\mathsf{path}(o_r, d_r)$ into a vector $\mathsf{vec}(o_r, d_r)$ with the labeled integers in a top-down manner in line 7. Since there are usually two edges at the same level, the edge closed to o_r appears before the edge closed to d_r. Finally, we obtain an ordered set of requests $\widehat{R^k}$ based on the lexicographic order of $\mathsf{vec}(o_r, d_r)$ in line 8. Here we use the bucket sort since each labeled integer can be viewed as a bucket and the length of $\mathsf{vec}(.,.)$ is no more than the height $\mathcal{H}$ of the HST.
(3) **Insertion.** In line 9, we apply the classic insertion procedure [46]. Specifically, we sequentially insert each request into the current route S_w based on the sorted order.

EXAMPLE 5. *Assume $R_{w_1} = \{r_1, r_2, r_3\}$ and $k = 3$ in the following example. In line 2, we know an HST is constructed as shown in Fig. 2b. In lines 3-4, we know $T' = T$ and $R^k = R_{w_1}$. In line 5, we then transform the paths of requests as in Example 4. Specifically, $\mathsf{path}(o_{r_1}, d_{r_1}) = \langle e_9, e_3, e_1, e_2, e_4, e_{10}\rangle$, $\mathsf{path}(o_{r_2}, d_{r_2}) = \langle e_{12}, e_5, e_4, e_{11}\rangle$ and $\mathsf{path}(o_{r_3}, d_{r_3}) = \langle e_{13}, e_6, e_7, e_{14}\rangle$. In line 6, the visiting order of edges is $\{e_8, e_9, e_3, e_1, e_2, e_4, e_{10}, e_{11}, e_5, e_{12}, e_6, e_{13}, e_7, e_{14}\}$. In line 7, we encode these paths by labelling e_8 as 0, e_9 as 1, etc. After that, we have $\mathsf{vec}(o_{r_1}, d_{r_1}) = \langle 3, 4, 2, 5, 1, 6\rangle$, $\mathsf{vec}(o_{r_2}, d_{r_2}) = \langle 8, 5, 9, 7\rangle$ and $\mathsf{vec}(o_{r_3}, d_{r_3}) = \langle 10, 12, 11, 13\rangle$. Then the lexicographic order is $\mathsf{vec}(o_{r_1}, d_{r_1}) < \mathsf{vec}(o_{r_2}, d_{r_2}) < \mathsf{vec}(o_{r_3}, d_{r_3})$ and hence $\widehat{R^k} = \{r_1, r_2, r_3\}$. In line 9, we first insert r_1 and obtain a route $\langle o_w, o_{r_1}, d_{r_1}\rangle$. We next insert r_2 and get a new route $\langle o_w, o_{r_1}, o_{r_2}, d_{r_1}, d_{r_2}\rangle$. Finally, we insert r_3 and obtain the route $S_w = \langle o_w, o_{r_1}, o_{r_2}, d_{r_1}, d_{r_2}, o_{r_3}, d_{r_3}\rangle$. Please refer to [46] for the detailed calculation procedure.*

Complexity Analysis. The time complexity of Algo. 3 is $O(\mathcal{H}n^2 + k^2)$ and its space complexity is $O(\mathcal{H}n)$, where $\mathcal{H}$ is the height of HST. Specifically, lines 1-2 take $O(\mathcal{H}n^2)$ to construct an HST. Lines 3-4 take $O(\mathcal{H}n)$ time to traverse the HST. Lines 5-7 take $O(\mathcal{H}k)$ time to transform the paths and encode them into vectors. Line 8 takes $O(\mathcal{H}k)$ time to implement the bucket sort. The sequential insertion in line 9 takes $O(k^2)$ time according to [57, 46].

Implementation. In practice, since the Algo. 3 is executed multiple times by the general framework, we can construct a global spatial index with all the locations. In this way, the time complexity of ESI improves to $O(\mathcal{H}n + k^2)$. Accordingly, the **time complexity** of the complete solution to the LMD problem is $O(\mathcal{H}n^2 + \mathcal{R}^3 m \log n + \mathcal{H}\mathcal{R}mn \log n)$ and its **space complexity** is $O(m + \mathcal{H}n)$, where $\mathcal{R}$ ($\ll n$) denotes the maximum number of requests assigned to the couriers and $\mathcal{H}$ denotes the height of the spatial index.

5.4 Approximation Analysis

We next prove the theoretical guarantee of ESI for the kLMD problem. Since HST is constructed by a randomized routine, ESI is intrinsically a randomized algorithm. To theoretically analyze a randomized algorithm, the expected value of the approximation ratio is a prevalent standard [54, 49, 17]. Specifically, the expected approximation ratio is the ratio of the expected result of the randomized algorithm to the optimal result in the worst-case. Our main result is summarized as follows.

THEOREM 2. *The (expected) approximation ratio of our algorithm ESI (Algo. 3) is $O(\log n)$.*

The **main idea** of proving Theorem 2 is:
(1) We first prove the lower bound in Lemma 4. Our lower bound is extended from [17], where the authors assume the weight of each request is a unit.
(2) We next prove the upper bound in Lemma 5.
(3) We finally prove Theorem 2 by Lemma 4 and Lemma 5.

For simplicity, we first introduce the additional **notations** in the following analysis. E_i denotes the set of edges on the HST at the i-th level. $R_{e,e'}$ denotes the set of requests whose paths $\mathsf{path}(.,.)$ go upward through edge e and go downward through edge e'. $y_{e,e'}$ denotes the total weight of requests in $R_{e,e'}$, *i.e.*, $y_{e,e'} = \sum_{r \in R_{e,e'}} c_r$.

Next, we extend the **lower bound** of the optimal solution in [17] to the setting of arbitrary weights.

LEMMA 4. *On the HST metric, the lower bound of the optimal solution is $\frac{1}{\sqrt{c_w}} \sum_{i=1}^{\mathcal{H}} \sum_{e,e' \in E_i} \left(\lceil \frac{y_{e,e'}}{c_w} \rceil \cdot |e|\right)$.*

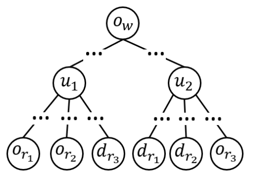
PROOF. We assume the optimal method has the power to split the delivered goods into unit pieces. For instance, a request with a weight c_r can be split into a set of c_r requests each of which has a unit weight. The above equation is the lower bound of such a powerful optimal solution [17]. However, the goods can not be split into pieces in practice. Thus, the actual optimal result is no smaller than it. □

Then we prove the **upper bound** of our algorithm ESI.

LEMMA 5. *On the HST metric, the upper bound of our algorithm ESI is $8 \sum_{i=1}^{\mathcal{H}} \sum_{e,e' \in E_i} \left(\lceil \frac{y_{e,e'}}{c_w} \rceil \cdot |e|\right)$.*

PROOF. The route of S_w consists of two parts, *i.e.*, the part that courier w has not picked up any requests and the part that courier w has already picked up the requests. Accordingly, the upper bound of ESI should be bounded by the sum of the total travel time of these two parts.

In the **first part**, let us first focus on the edges E_i at the i-th level. For any two different edges $e, e' \in E_i$, there



(a) An example of HST with $\{r_1, r_2, r_3\}$.

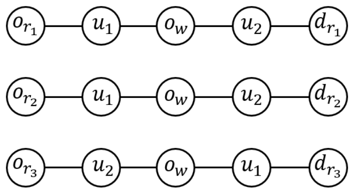

o_{r_1} u_1 o_w u_2 d_{r_1}
o_{r_2} u_1 o_w u_2 d_{r_2}
o_{r_3} u_2 o_w u_1 d_{r_3}
(b) the paths of $\{r_1, r_2, r_3\}$ on the HST.

Figure 3: An example of the HST and the paths.

are totally $y_{e,e'}$ requests which need to go upward through edge e and then go downward through edge e'. Since these requests have already shared the edges e, e', they must also share the same edges at the higher levels. Thus, after the sorting step (line 8) in Algo. 3, these requests will be in adjacent positions in the sorted order $\widehat{R^k}$. To satisfy the capacity constraint, the courier has to traverse the edges e, e' with no picked requests for $2\lceil\frac{y_{e,e'}}{c_w}\rceil$ times at most. This is because at least half of the capacity c_w can be fully loaded when the weights of requests are not unit, *i.e.*, $\lceil\frac{y_{e,e'}}{0.5c_w}\rceil \le 2\lceil\frac{y_{e,e'}}{c_w}\rceil$. As the edges at the same level (*e.g.*, e, e') have the same length, the total travel time of this part is bounded by

$$\sum_{i=1}^{\mathcal{H}}\sum_{e,e'\in E_i}\left(2\lceil\frac{y_{e,e'}}{c_w}\rceil\cdot 2|e|\right) = 4\sum_{i=1}^{\mathcal{H}}\sum_{e,e'\in E_i}\left(\lceil\frac{y_{e,e'}}{c_w}\rceil\cdot|e|\right) \quad (5)$$

In the **second part**, we prove the upper bound in the following two cases.

Case 1. We first focus only on the set of requests (denoted by $R_{e,e'}$) in $y_{e,e'}$. As shown in Fig. 3a, we assume the requests $\{r_1, r_2\}$ belong to $R_{e,e'}$ as they have common edges from u_1 to u_2 (see Fig. 3b). Suppose the courier is currently located at the root o_w. Fig. 4a illustrates the route after sequentially inserting r_1, r_2 on the HST, *i.e.*,

$$\langle o_w, (u_1,)\ o_{r_1}, (u_1,)\ o_{r_2}, (u_1, o_w, u_2,)\ d_{r_1}, (u_2,)\ d_{r_2}\rangle$$

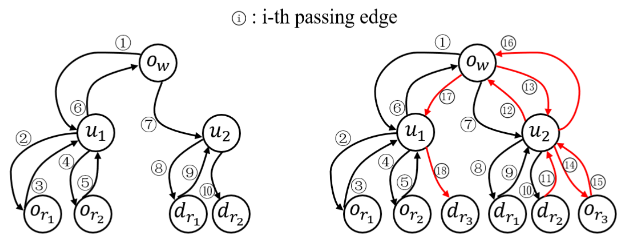
As the insertion operator only outputs the sequences of origins and destinations, we use brackets to represent the nodes that the route passes through. Obviously, the detour of insertion only happens at the branches of non-shared parts. Thus, the travel time of the shared edges e, e' are equally amortized by the requests $R_{e,e'}$. As explained above, the courier has to traverse the edges e, e' to deliver the requests for $2\lceil\frac{y_{e,e'}}{c_w}\rceil$ times. As the edges at the same level (*e.g.*, e and e') have the same length on the HST, the total travel time of this part is bounded by

$$\sum_{i=1}^{\mathcal{H}}\sum_{e,e'\in E_i}\left(2\lceil\frac{y_{e,e'}}{c_w}\rceil\cdot 2|e|\right) = 4\sum_{i=1}^{\mathcal{H}}\sum_{e,e'\in E_i}\left(\lceil\frac{y_{e,e'}}{c_w}\rceil\cdot|e|\right) \quad (6)$$

Case 2. In some cases, after inserting all the requests $R_{e,e'}$, we will insert the requests that have no shared parts with $R_{e,e'}$. For example, the request r_3 in Fig. 3 is one of such requests. We want to prove that the upper bound Eq. (5) still holds after inserting r_3. As shown in Fig. 4b, there is always a possible insertion plan to place o_{r_3} at the end of the route, *i.e.*, the courier first delivers all the requests $R_{e,e'}$ and then the new request outside $R_{e,e'}$ (*i.e.*, red edges). Therefore, the travel time of such a simple route is still bounded by Eq. (6). Since the *insertion* operator can get the route with the shortest travel time after adding the new request, it should also be bounded by Eq. (6).

The upper bound is derived by the sum of Eq. (5)-(6). □

We finally prove the **ratio** of algorithm ESI in Theorem 2.



(a) the route after inserting $\{r_1, r_2\}$.
(b) the route after inserting $\{r_1, r_2, r_3\}$.

Figure 4: An example of inserting requests $\{r_1, r_2, r_3\}$.

Proof. Let $D_T(S_w^*)$ and $D(S_w^*)$ be the minimum total travel time following the optimal route S_w^* on the HST metric and original metric respectively. Let $D_T(S_w)$ and $D(S_w)$ be the total travel time obtained by our algorithm ESI on the HST metric and original metric respectively. We also use $\mathcal{C}$ to denote the ratio between the longest requests and the shortest requests, *i.e.*, $\mathcal{C} = \max_r \mathsf{d}(o_r, d_r)/\min_r \mathsf{d}(o_r, d_r)$. In the following, $\mathcal{C}$ is assumed to be a constant, which is practical in the last-mile delivery platforms [30, 57, 18]. For example, [30] provides the spatial distribution of 1,852,439 food delivery orders collected in Shanghai. The delivery distances of all these requests are strictly less than 4000 meters while the delivery distances of these requests are over 100 meters. In this case, $\mathcal{C}$ can be viewed as 40.

Thus, in the worst case, the k requests selected by ESI are the longest while the k requests selected by the optimal solution are the shortest. Based on Lemma 4 and Lemma 5, we can derive the ratio of ESI on the HST as

$$\frac{D_T(S_w)}{D_T(S_w^*)} \le \mathcal{C}\cdot\frac{\frac{1}{\sqrt{c_w}}\sum_{i=1}^{\mathcal{H}}\sum_{e,e'\in E_i}\left(\lceil\frac{y_{e,e'}}{c_w}\rceil\cdot|e|\right)}{8\sum_{i=1}^{\mathcal{H}}\sum_{e,e'\in E_i}\left(\lceil\frac{y_{e,e'}}{c_w}\rceil\cdot|e|\right)} = O(\sqrt{c_w}).$$

Based on the first property of the HST (see Sec. 5.1.1), we know $\mathsf{d}_T(u,v) \ge \mathsf{d}(u,v)$ and $\mathbb{E}[\mathsf{d}_T(u,v)] \ge O(\log n)\cdot\mathsf{d}(u,v)$. Thus, we have $\mathbb{E}[D_T(S_w^*)] \ge O(\log n)\cdot D(S_w^*)$ and $D_T(S_w) \ge D(S_w)$. Finally, since the capacity c_w is a small constant ($\ll n, m$) in practice, the (expected) approximation ratio is

$$\text{ratio} = \frac{\mathbb{E}[D(S_w)]}{D(S_w^*)} \le O(\log n)\cdot\mathbb{E}\left[\frac{D_T(S_w)}{D_T(S_w^*)}\right] = O(\log n).$$

□

5.5 Discussions

In practice, last-mile delivery can be applied in many scenarios with different settings. In the following, we discuss how to extend our algorithm to the online scenario and the scenario with deadline constraint.

(1) **Online scenario.** In the online scenario, the requests dynamically arrive at the platform. To plan the routes in the online scenario, we can apply the widely-used batch-based mode [60, 53, 47]. The main idea is to determine the length of a time period and then plan the part of routes for the set of requests which have arrived in this time period. Iteratively, all the requests will be completed. Thus, in each batch, our algorithm can be used to plan the routes for the available couriers to complete this batch of requests. Moreover, our theoretical guarantees still hold for the batch-based mode. Please refer to our full paper [6] for the proof.

(2) **Deadline constraint.** Some of the requests may have preferred deadline constraints (*i.e.*, the latest time to reach origin and destination). For instance, in the real datasets collected by Cainiao [4], 8,856 out of 238,636 the

packages (3.71%) have these deadlines, and all the other packages (96.29%) do not any deadlines. In other word, the number of such requests is small and most of the requests have no deadlines for either pickup or delivery. To address this scenario, we first use our algorithm to plan the routes for the requests without the deadlines. After that, we use the linear-time insertion algorithm [46] to sequentially insert the requests with deadlines into the current routes. Since more constraint is involved, the theoretical guarantees do not hold in this extension.

6. EXPERIMENTAL STUDY

6.1 Experimental Setup

Synthetic Datasets. We generate the synthetic datasets based on the parameter settings in Table 3, where the default parameters are marked in bold. Specifically, we vary the number of requests $|R|$, the number of couriers $|W|$, etc. Similar to [57], weights are generated by Gaussian distribution whose mean value varies from 1 to 3. To generate the locations, we apply the widely used method [43, 53, 44, 45, 35, 40, 51, 52]. Specifically, we generate the initial locations of couriers and the origins and destinations of requests on a two-dimensional Euclidean space with size 200×200 under the Uniform, Gaussian, and Exponential distributions. For the scalability tests, we generate the locations on a larger Euclidean space with size 500×500.

Real Datasets. As shown in Table 4, we use the two publicly accessed real datasets of last-mile delivery, *i.e.*,*Cainiao* and *Olist*. The *Cainiao* dataset [4] is collected in Shanghai by the largest logistics platform called Cainiao [3] owned by Alibaba Group [1]. The requests in *Cainiao* include both food delivery requests and parcel delivery requests. We use the same procedure to process the raw data as [57]. We also use a smaller dataset *Olist*, which is generated by the public datasets from Kaggle [10], It includes the real-world e-commerce requests of a Brazilian e-commerce company called Olist [9]. We extract the requests whose origins and destinations are both in the city of Sao Paulo, because Sao Paulo involves the maximum number of such requests. The two real datasets include all the information except the initial locations of couriers. Therefore, we randomly generate the initial locations in the spatial range of the requests and vary the number of couriers in Table 4.

Metrics and Baselines. We compare our method (named by FESI) with the following baselines in terms of makespan of couriers (makespan for short), total latency of requesters (total latency for short), total travel time of couriers (total travel time for short), running time and memory usage.

(1) pruneGreedyDP [46] (GDP for short) devises the most efficient insertion operation to minimize the total travel time.

(2) TAC [50] is a TSP based solution to minimize the latency of couriers. Its basic idea is to iteratively select a set of requests and then combine the TSP tour of their origins and the other TSP tour of their destinations.

(3) [17] achieve the best-known approximation ratio in minimizing the makespan when there is one courier. We extend their algorithm (denoted by eFOCS) to the case of multiple couriers by the idea of hot spots [29]. Specifically, we use binary-search to decide the radius of the hot spot for each courier. Then, a courier is assigned to the set of requests in his/her hot spot. Thus, their method is used to plan the route for each courier to complete his/her requests.

Table 3: Synthetic datasets.

$\|R\|$	2000,4000,**6000**,8000,10000
$\|W\|$	30,60,**100**,120,150
weight c_r	1,1.5,**2**,2.5,3
capacity c_w	4,6,**8**,10,15
location distributions	*mean* of Uniform: 50,75,100,125,150 μ of **Gaussian**: 50,75,**100**,125,150 σ of **Gaussian**: 5,10,**15**,20,25 λ of Exponential: 50,75,100,125,150
scalability ($\|R\|,\|W\|$)	$(20k, 1k)$, $(40k, 2k)$, $\cdots$, $(200k, 10k)$

Table 4: Real datasets.

Dataset	$\|R\|$	$\|W\|$	Collected City
Cainiao	238,636	1k,2k,3k,4k,5k	Shanghai, China
Olist	4,819	30,60,100,120,150	Sao Paulo, Brazil

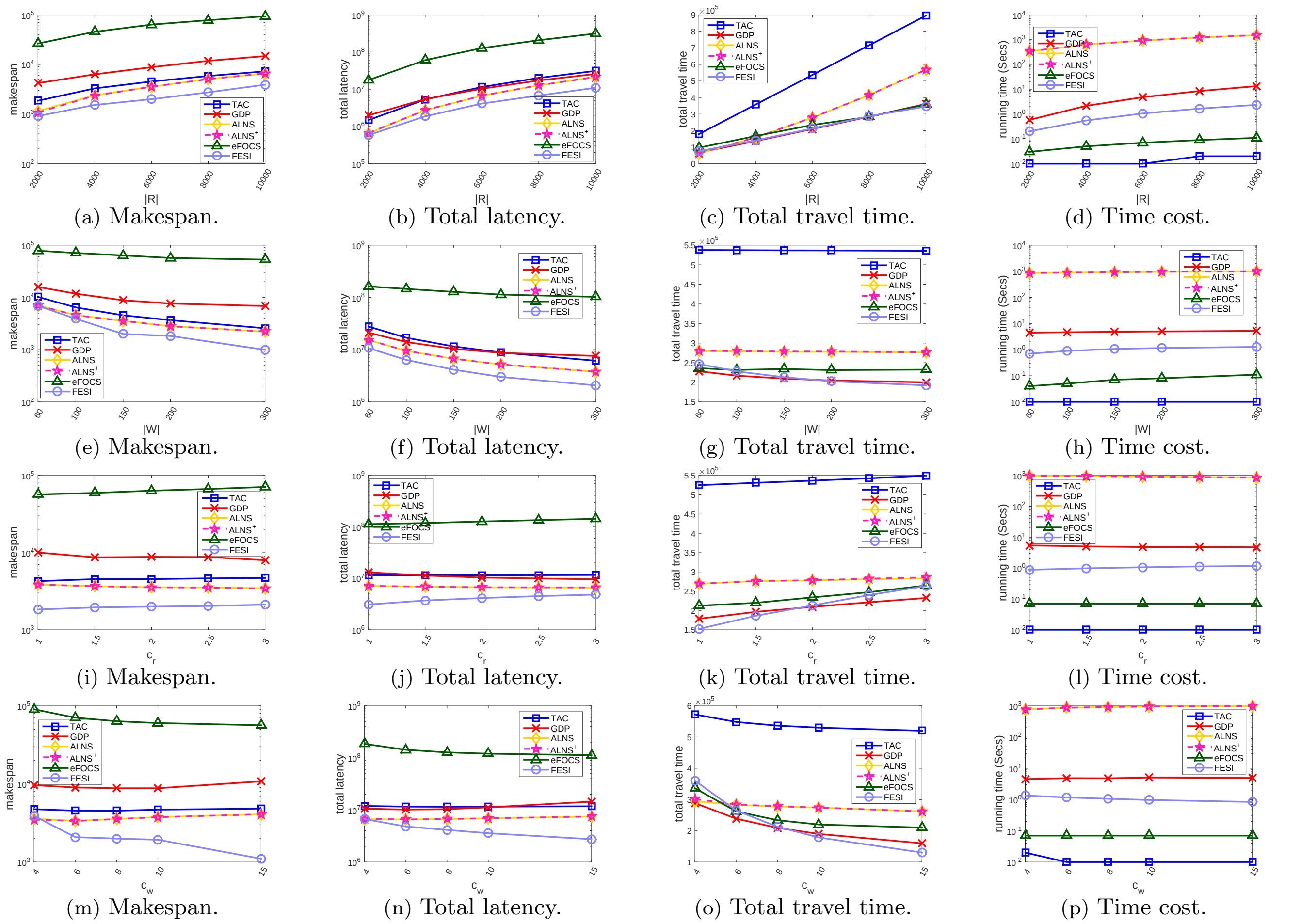
(4) Adaptive large neighborhood search (ALNS) is one of the widely used algorithms from the area of transportation science. Gschwind and Drexl [25] propose the state-of-the-art ALNS algorithm (denoted by ALNS), which is viewed as the most effective and efficient ALNS algorithm to minimize the total travel time of couriers in a recent survey [28]. We optimize ALNS by the linear-time insertion operation in [46]. Moreover, we also use a weighted sum of the two objectives in our LMD problem to extend the algorithm (denoted by ALNS^+), *i.e.*, $\text{OBJ}_1 + \frac{1}{n}\text{OBJ}_2$, since most related studies (see Table 5 in the survey [28]) with multiple objectives use a weighted sum of the objectives as the optimization goal.

Though some baselines are designed based on a single objective, we still conduct experiments as it is unknown whether they are effective in the other practical objectives.

Implementation. We implement all the compared algorithms in GNU C++. The experiments are conducted on a server with 40 Intel(R) Xeon(R) E5 2.30GHz processors and 128GB memory. Each experiment is repeated 50 times and the average results are reported. Due to space limitations, we report the memory usage in Sec. 6.2 and refer readers to our full paper [6] for more illustrations.

6.2 Experimental Results

Impact of the number of requests. The first row of Fig. 5 shows the experimental results of varying the number of requests. In terms of both objectives (*i.e.*, makespan and total latency), our algorithm FESI outperforms other baselines. Specifically, FESI has up to 45.9%, 45.9%, 56.0%, 78.3% and 96.8% shorter makespan than ALNS, ALNS^+, TAC, GDP and eFOCS respectively. Simultaneously, FESI has up to 48.7%, 48.7%, 66.3%, 70.6% and 96.9% lower total latency than ALNS, ALNS^+, TAC, GDP and eFOCS respectively. Among the baselines, ALNS^+ is only a little better than ALNS and both of them are more effective than the other baselines. GDP has longer makespan and lower total latency than TAC although TAC instead of GDP focuses on minimizing the total latency. eFOCS is always the least effective in these two objectives but it is comparable in minimizing the total travel time. As shown in Fig. 5c, eFOCS has averagely 55.4% shorter total travel time than TAC, which is the least effective in this metric. Our method FESI takes up to 36.9%, 36.9%, 60.6% and 21.9% shorter total travel time than ALNS, ALNS^+, TAC and eFOCS respectively. Besides, ALNS is the most effective when $|R|$ is 2k and eFOCS is the most effective when $|R|$ is 10k. In terms of running time, ALNS and ALNS^+ are the slowest, TAC is the fastest and others are efficient enough. Particularly, ALNS and ALNS^+ take up to 92257×, 581×, 13796× and 1689× longer running



(a) Makespan. (b) Total latency. (c) Total travel time. (d) Time cost.

(e) Makespan. (f) Total latency. (g) Total travel time. (h) Time cost.

(i) Makespan. (j) Total latency. (k) Total travel time. (l) Time cost.

(m) Makespan. (n) Total latency. (o) Total travel time. (p) Time cost.

Figure 5: Results on varying $|R|$, $|W|$, weight c_r and capacity c_w on synthetic datasets.

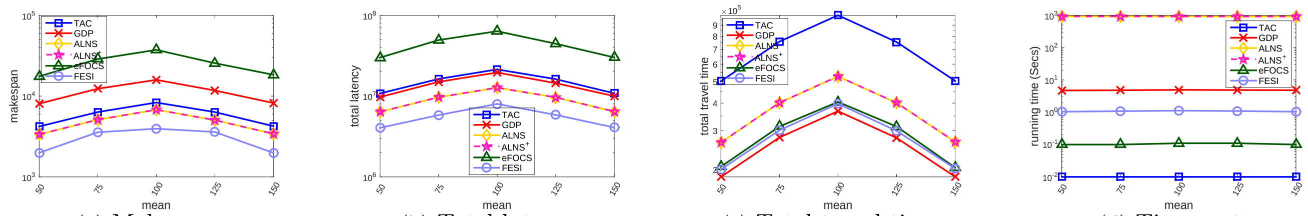
time than TAC, GDP, eFOCS and FESI respectively. In terms of memory usage, all the algorithms are efficient enough to consume less than 17MB.

Impact of the number of couriers. The second row of Fig. 5 shows the experimental results of varying the number of couriers. With the increase of W, the makespan and total latency of all algorithms decrease. The rankings of all algorithms in terms of both makespan and total latency is still the same as previous results. Our algorithm FESI still outperforms these baselines with 30.1%, 30.0%, 47.8%, 72.4%, 95.4% shorter makespan and 37.7%, 37.8%, 63.9%, 60.4%, 96.2% lower total latency than ALNS, ALNS$^+$, TAC, GDP and eFOCS in average. As for total travel time, ALNS, ALNS$^+$ and TAC are notably longer than the others. When $|W|$ is smaller than 150, GDP has the shortest total travel time. However, when $|W|$ increases over 200, FESI becomes the most effective. eFOCS is often in the third place with averagely 10.2% and 8.4% longer total travel time than GDP and FESI. As for running time, TAC is the most efficient while ALNS and ALNS$^+$ are the least efficient (789×-1228× slower than FESI). Besides, all the algorithms are efficient in memory usage.

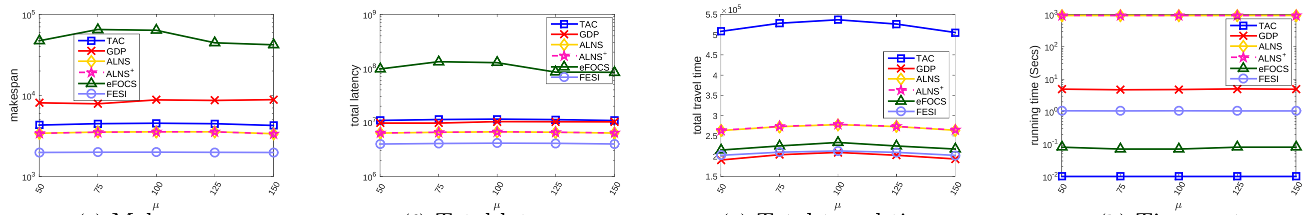
Impact of the weight of the requests. The third row of Fig. 5 shows the experimental results of varying the weight of requests. Our method FESI is still the best in terms of both makespan and total latency, followed by ALNS$^+$, ALNS, TAC, GDP and eFOCS. As for total travel time, FESI is less effective than GDP when the weight is large. However, when the weight is small, FESI takes the shortest total travel time by 14.8%-28.3% than GDP and eFOCS. The other baselines still have notably longer total travel time. In terms of time cost and memory usage, FESI is still efficient.

Impact of the capacity of the couriers. The last row of Fig. 5 shows the experimental results of varying the capacity of couriers. For both makespan and total latency, we observe similar patterns to previous results. In terms of total travel time, FESI has longer total travel time than ALNS, ALNS$^+$, GDP and eFOCS when the capacity is small. However, when the capacity increases, it eventually becomes the most effective algorithm. For running time, ALNS and ALNS$^+$ are still 169×-99087× slower than the others. The memory usage is all less than 16MB.

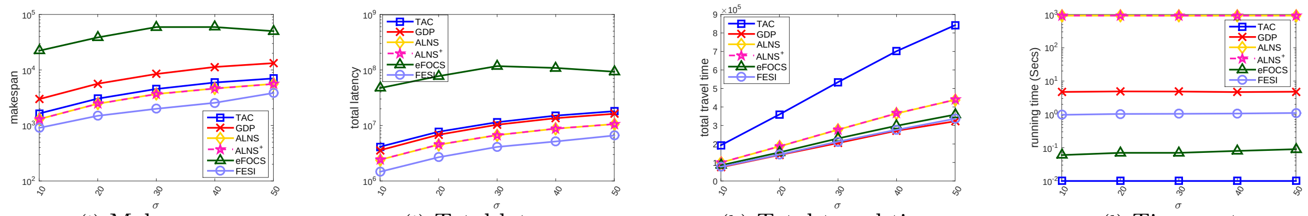
Impact of the mean of Uniform distribution. The first row of Fig. 6 shows the experimental results when locations follow the Uniform distribution with the parameter *mean*. In Fig. 6a-Fig. 6c, the results are all first increasing and then decreasing. The reason is the spatial range when $mean = 50$ is smaller than the spatial range when $mean = 100$. The locations in the former case fall into an area with size 100×100 instead of 200×200 in the latter case. In the Uniform distribution, we can still observe that FESI outperforms the others in terms of both makespan and total latency. As for total travel time, we see a similar pattern to previous results. As for time and memory cost, the rankings of these



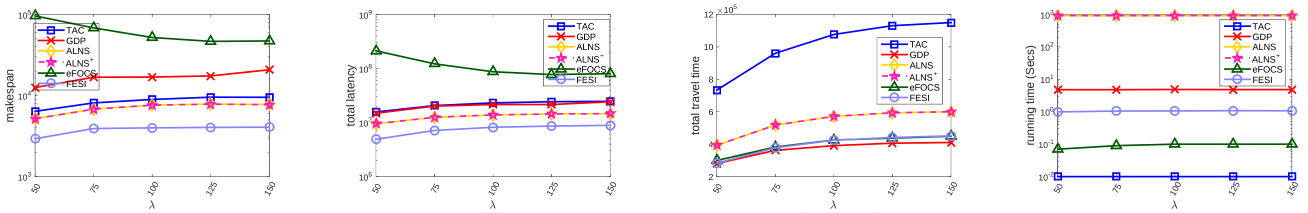
(a) Makespan. (b) Total latency. (c) Total travel time. (d) Time cost.



(e) Makespan. (f) Total latency. (g) Total travel time. (h) Time cost.



(i) Makespan. (j) Total latency. (k) Total travel time. (l) Time cost.



(m) Makespan. (n) Total latency. (o) Total travel time. (p) Time cost.

Figure 6: Results on varying Uniform (*mean*), Gaussian (μ, σ) and Exponential (λ) distributions on synthetic datasets.

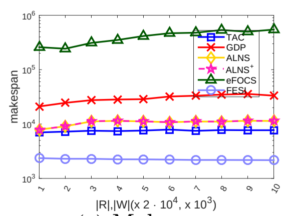
algorithms are also similar to previous results.

Impact of the μ and σ of Gaussian distribution. The second and third rows of Fig. 6 show the experimental results when locations follow the Gaussian distribution with the parameter μ, σ. We can observe all algorithms are sensitive to σ and eFOCS also is sensitive to μ. In terms of makespan and total latency, FESI is always the best and ALNS and ALNS$^+$ are the runners-up. In terms of total travel time, GDP is the best and FESI is the runner-up. The gaps between them are still small. In terms of running time, ALNS and ALNS$^+$ are the slowest and TAC is the fastest. As for memory cost, it is no more than 16MB.

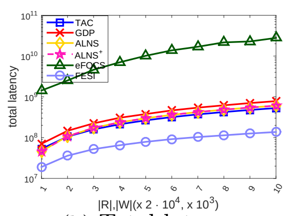
Impact of the λ of Exponential distribution. The last row of Fig. 6 shows the experimental results when locations follow the Exponential distribution with the parameter λ. We can observe the makespan and total latency of eFOCS decrease with the increase of λ, while for the other algorithms, the values of these two objectives are relatively stable. In the Exponential distribution, FESI is still the best. It obtains at least 42.3% shorter makespan and at least 39.1% lower total latency than all the baselines. As for total travel time, TAC is still the least effective, followed by ALNS$^+$, ALNS, eFOCS, FESI and GDP. As for time cost and memory cost, the pattern is similar to previous results.

Impact on the scalability tests. The first row of Fig. 7 shows the experimental results on scalability tests. ALNS and ALNS$^+$ usually cannot iterate enough times (*e.g.*, less than 20 when $|R| = 200k$) in the scalability tests. We only report their best results within 2 hours. FESI always outperforms the others in terms of makespan, total latency and total travel time. Specifically, FESI has at least 70.0%, 70.0%, 66.2%, 88.7% and 99.0% shorter makespan and 58.9%, 58.8%, 64.0%, 73.0% and 98.5% lower total latency than ALNS, ALNS$^+$, TAC, GDP and eFOCS respectively. Besides, FESI has averagely 70.2%, 70.3%, 67.9%, 4.3% and 8.7% shorter total travel time than these baselines. In terms of running time, TAC and eFOCS are notably efficient than others. Our FESI algorithm is the third most efficient algorithm while GDP, ALNS and ALNS$^+$ consume up to 2×, 96× and 96× more time respectively.

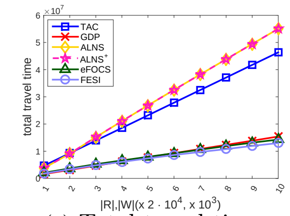
Impact on the real datasets. The last two rows of Fig. 7 show the experimental results on real datasets. In *Olist* (Fig. 7e-Fig. 7h), the rankings of algorithms in terms of makespan and total latency is similar to the results in synthetic datasets. As for total travel time, GDP is the most effective and FESI is the runner-up. In *Cainiao*, our algorithm FESI notably outperforms all the baselines in terms of makespan, total latency and total travel time (Fig. 7i-Fig. 7k). Specifically, the compared baselines have 29.3×-108.9× longer makespan and 20.2×-175.1× higher total latency than our method FESI. Even for total travel time, they are still worse than our method FESI by 1.6×-122.2×. In terms of efficiency, ALNS and ALNS$^+$ are inefficient in



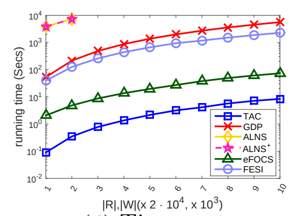
(a) Makespan.



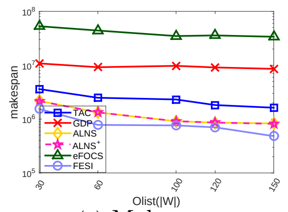
(b) Total latency.



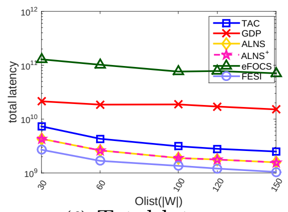
(c) Total travel time.



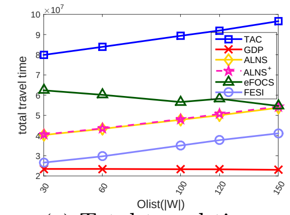
(d) Time cost.



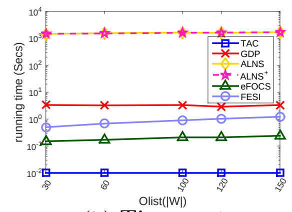
(e) Makespan.



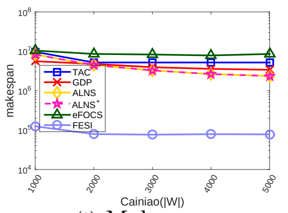
(f) Total latency.



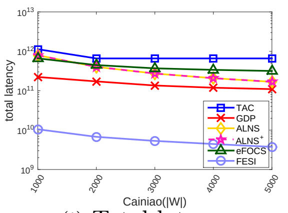
(g) Total travel time.



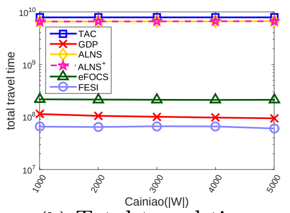
(h) Time cost.



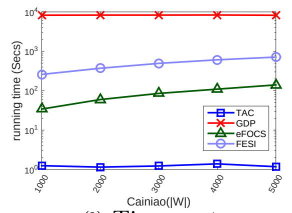
(i) Makespan.



(j) Total latency.



(k) Total travel time.



(l) Time cost.

Figure 7: Results on scalability tests and real datasets.

real datasets. For example, they consume 430×-168218× longer time than other algorithms in *Olist*. In large-scale real dataset (*Cainiao*), ALNS and ALNS$^+$ fail to terminate in 2 hours and execute less than 20 iterations while TAC, eFOCS and FESI can be terminated in less than 12 minutes. TAC is the most efficient and GDP consumes notably longer time. Moreover, the gaps among FESI, eFOCS and TAC are much smaller than the results on scalability tests. Since *Cainiao* is a large-scale dataset with over 200k requests, the results demonstrate the efficiency of FESI. As for memory usage, all the algorithms consume less than 95MB space.

Summary. Our experimental findings are summarized as:

- In terms of makespan and total latency, ALNS and ALNS$^+$ are comparably effective but they are also notably inefficient. Other efficient baselines are either ineffective in these two objectives (GDP and eFOCS) or sacrifice notably longer total travel time (TAC).
- Our method FESI is usually the most effective algorithm with averagely 48.4%-96.0% and 49.7%-96.1% higher improvements in terms of makespan and total latency than all the baselines. Simultaneously, FESI only has averagely 2.5% higher total travel time than GDP but obtains averagely 15.5%-64.7% shorter total travel time than the other baselines. Especially in the large-scale real dataset (*Cainiao*), FESI obtains 29.3×-108.9× shorter makespan and 20.2×-175.1× lower total latency with shorter total travel time than the state-of-the-art algorithms with good efficiency.
- Among the baselines, ALNS and ALNS$^+$ are comparably effective in makespan and total latency. However, they are 112×-168218× slower than the other baselines (when $n \leq 10k$). TAC is the most efficient while it is notably ineffective in the large-scale real dataset. eFOCS is also efficient but ineffective in minimizing both makespan and total latency. GDP is often comparably effective in terms of total latency and total travel time while it also has notably longer makespan.

7. CONCLUSION

In this paper, we propose the LMD problem, which plans the routes among the couriers and requesters in last-mile delivery for two objectives, *i.e.*, minimizing the makespan of couriers and total latency of requesters. The problem with either objective is strongly NP-hard and hence we focus on designing an efficient method with theoretical guarantees. Specifically, we propose a framework with both approximation ratios of 6ρ, where ρ is the approximation ratio of its core operation called kLMD. We next apply the spatial index called hierarchically separated tree and devise an approximation algorithm ESI for the kLMD problem with $\rho = O(\log n)$, where n is the number of requests. Finally, extensive experiments on both synthetic and real datasets show that our approach outperforms the state-of-the-art algorithms in terms of both makespan and total latency with averagely 48.4%-96.0% and 49.7%-96.1% improvements.

8. ACKNOWLEDGMENTS

We thank the reviewers for their constructive comments. Yuxiang Zeng and Lei Chen's works are partially supported by the Hong Kong RGC GRF Project 16209519, the National Science Foundation of China (NSFC) under Grant No. 61729201, Science and Technology Planning Project of Guangdong Province, China, No. 2015B010110006, Hong Kong ITC ITF grants ITS/044/18FX and ITS/470/18FX, Didi-HKUST joint research lab Grant, Microsoft Research Asia Collaborative Research Grant and Wechat Research Grant. Yongxin Tong's work is partially supported by the National Science Foundation of China (NSFC) under Grant No. 61822201 and U1811463. Yongxin Tong and Lei Chen are the corresponding authors in this paper.

9. REFERENCES

[1] Alibaba Group. *https://www.alibabagroup.com/*, 2019.

[2] Amazon. *https://www.amazon.com/*, 2019.

[3] Cainiao. *https://www.cainiao.com/*, 2019.

[4] Cainiao dataset published in Tianchi. *https://tianchi.aliyun.com/competition/entrance/231-581/introduction*, 2019.

[5] FedEx. *http://www.fedex.com/*, 2019.

[6] Full paper. *http://home.cse.ust.hk/~yzengal/fesi.pdf*, 2019.

[7] Labour Law of the People's Republic of China. *https://en.wikipedia.org/wiki/Labour_Law_of_the_People's_Republic_of_China*, 2019.

[8] Meituan. *http://www.meituan.com/*, 2019.

[9] Olist. *https://www.crunchbase.com/organization/olist*, 2019.

[10] Olist dataset published in Kaggle. *https://www.kaggle.com/olistbr/brazilian-ecommerce*, 2019.

[11] Seamless. *http://www.seamless.com/*, 2019.

[12] Strong NP-completeness. *https://en.wikipedia.org/wiki/Strong_NP-completeness*, 2019.

[13] The road to the world champion: the development of trip planning engine for the vehicle routing problem in Cainiao (Chinese). *https://yq.aliyun.com/articles/697250*, 2019.

[14] United States labor law. *https://en.wikipedia.org/wiki/United_States_labor_law*, 2019.

[15] Y. Bartal. Probabilistic approximations of metric spaces and its algorithmic applications. In *FOCS*, pages 184–193, 1996.

[16] X. Bei and S. Zhang. Algorithms for trip-vehicle assignment in ride-sharing. In *AAAI*, pages 3–9, 2018.

[17] M. Charikar and B. Raghavachari. The finite capacity dial-a-ride problem. In *FOCS*, pages 458–467, 1998.

[18] C. Chen, D. Zhang, L. Wang, X. Ma, X. Han, and E. H. Sha. Taxi exp: A novel framework for city-wide package express shipping via taxi crowd sourcing. In *UIC/ATC/ScalCom*, pages 244–251, 2014.

[19] L. Chen, Q. Zhong, X. Xiao, Y. Gao, P. Jin, and C. S. Jensen. Price-and-time-aware dynamic ridesharing. In *ICDE*, pages 1061–1072, 2018.

[20] Z. Chen, P. Cheng, Y. Zeng, and L. Chen. Minimizing maximum delay of task assignment in spatial crowdsourcing. In *ICDE*, pages 1454–1465, 2019.

[21] A. Das, S. Gollapudi, A. Kim, D. Panigrahi, and C. Swamy. Minimizing latency in online ride and delivery services. In *WWW*, pages 379–388, 2018.

[22] W. de Paepe, J. K. Lenstra, J. Sgall, R. A. Sitters, and L. Stougie. Computer-aided complexity classification of dial-a-ride problems. *INFORMS Journal on Computing*, 16(2):120–132, 2004.

[23] J. Fakcharoenphol, S. Rao, and K. Talwar. A tight bound on approximating arbitrary metrics by tree metrics. In *STOC*, pages 448–455, 2003.

[24] S. Gollapudi and D. Sivakumar. Framework and algorithms for trend analysis in massive temporal data sets. In *CIKM*, pages 168–177, 2004.

[25] T. Gschwind and M. Drexl. Adaptive large neighborhood search with a constant-time feasibility test for the dial-a-ride problem. *Transportation Science*, 53(2):480–491, 2019.

[26] S. R. B. Gummidi, X. Xie, and T. B. Pedersen. A survey of spatial crowdsourcing. *ACM Transactions on Database Systems*, 44(2):8:1–8:46, 2019.

[27] A. Gupta, M. T. Hajiaghayi, V. Nagarajan, and R. Ravi. Dial a ride from k-forest. *ACM Transactions on Algorithms*, 6(2):41:1–41:21, 2010.

[28] S. C. Ho, W. Y. Szeto, Y. H. Kuo, J. M. Y. Leung, M. Petering, and T. W. H. Tou. A survey of dial-a-ride problems: Literature review and recent developments. *Transportation Research Part B Methodological*, 111, 2018.

[29] Y. Huang, F. Bastani, R. Jin, and X. S. Wang. Large scale real-time ridesharing with service guarantee on road networks. *PVLDB*, 7(14):2017–2028, 2014.

[30] S. Ji, Y. Zheng, Z. Wang, and T. Li. Alleviating users' pain of waiting: Effective task grouping for online-to-offline food delivery services. In *WWW*, pages 773–783, 2019.

[31] F. Li, D. Cheng, M. Hadjieleftheriou, G. Kollios, and S. Teng. On trip planning queries in spatial databases. In *SSTD*, pages 273–290, 2005.

[32] J. Li, A. Deshpande, and S. Khuller. Minimizing communication cost in distributed multi-query processing. In *ICDE*, pages 772–783, 2009.

[33] M. Li, J. Zhang, and W. Wang. Task selection and scheduling for food delivery: A game-theoretic approach. In *GLOBECOM*, pages 1–6, 2018.

[34] Y. Li, D. Deng, U. Demiryurek, C. Shahabi, and S. Ravada. Towards fast and accurate solutions to vehicle routing in a large-scale and dynamic environment. In *SSTD*, pages 119–136, 2015.

[35] Y. Li, J. Fang, Y. Zeng, B. Maag, Y. Tong, and L. Zhang. Two-sided online bipartite matching in spatial data: experiments and analysis. *GeoInformatica*, 2019.

[36] Y. Liu, B. Guo, C. Chen, H. Du, Z. Yu, D. Zhang, and H. Ma. Foodnet: Toward an optimized food delivery network based on spatial crowdsourcing. *IEEE Transactions on Mobile Computing*, 18(6):1288–1301, 2019.

[37] S. Ma, Y. Zheng, and O. Wolfson. T-share: A large-scale dynamic taxi ridesharing service. In *ICDE*, pages 410–421, 2013.

[38] S. Ma, Y. Zheng, and O. Wolfson. Real-time city-scale taxi ridesharing. *IEEE Transactions on Knowledge and Data Engineering*, 27(7):1782–1795, 2015.

[39] S. Ropke and D. Pisinger. An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows. *Transportation Science*, 40(4):455–472, 2006.

[40] Q. Tao, Y. Zeng, Z. Zhou, Y. Tong, L. Chen, and K. Xu. Multi-worker-aware task planning in real-time spatial crowdsourcing. In *DASFAA*, pages 301–317, 2018.

[41] R. S. Thangaraj, K. Mukherjee, G. Raravi, A. Metrewar, N. Annamaneni, and K. Chattopadhyay. Xhare-a-ride: A search optimized dynamic ride

sharing system with approximation guarantee. In *ICDE*, pages 1117–1128, 2017.

[42] Y. Tong, L. Chen, and C. Shahabi. Spatial crowdsourcing: Challenges, techniques, and applications. *PVLDB*, 10(12):1988–1991, 2017.

[43] Y. Tong, J. She, B. Ding, L. Chen, T. Wo, and K. Xu. Online minimum matching in real-time spatial data: Experiments and analysis. *PVLDB*, 9(12):1053–1064, 2016.

[44] Y. Tong, J. She, B. Ding, L. Wang, and L. Chen. Online mobile micro-task allocation in spatial crowdsourcing. In *ICDE*, pages 49–60, 2016.

[45] Y. Tong, L. Wang, Z. Zhou, B. Ding, L. Chen, J. Ye, and K. Xu. Flexible online task assignment in real-time spatial data. *PVLDB*, 10(11):1334–1345, 2017.

[46] Y. Tong, Y. Zeng, Z. Zhou, L. Chen, J. Ye, and K. Xu. A unified approach to route planning for shared mobility. *PVLDB*, 11(11):1633–1646, 2018.

[47] Y. Tong and Z. Zhou. Dynamic task assignment in spatial crowdsourcing. *SIGSPATIAL Special*, 10(2):18–25, 2018.

[48] Y. Tong, Z. Zhou, Y. Zeng, L. Chen, and C. Shahabi. Spatial crowdsourcing: a survey. *The VLDB Journal*, 2019.

[49] V. V. Vazirani. *Approximation algorithms*. Springer, 2001.

[50] H. A. Waisanen, D. Shah, and M. A. Dahleh. A dynamic pickup and delivery problem in mobile networks under information constraints. *IEEE Transactions on Automatic Control*, 53(6):1419–1433, 2008.

[51] Y. Wang, L. Chen, Y. Che, and Q. Luo. Accelerating pairwise simrank estimation over static and dynamic graphs. *The VLDB Journal*, 28(1):99–122, 2019.

[52] Y. Wang, X. Lian, and L. Chen. Efficient simrank tracking in dynamic graphs. In *ICDE*, pages 545–556, 2018.

[53] Y. Wang, Y. Tong, C. Long, P. Xu, K. Xu, and W. Lv. Adaptive dynamic bipartite graph matching: A reinforcement learning approach. In *ICDE*, pages 1478–1489, 2019.

[54] D. P. Williamson and D. B. Shmoys. *The Design of Approximation Algorithms*. Cambridge University Press, 2011.

[55] N. H. Wilson, R. Weissberg, B. Higonnet, and J. Hauser. Advanced dial-a-ride algorithms. Technical report, 1975.

[56] H. Xia and H. Yang. Is last-mile delivery a 'killer app' for self-driving vehicles? *Communications of the ACM*, 61(11):70–75, 2018.

[57] Y. Xu, Y. Tong, Y. Shi, Q. Tao, K. Xu, and W. Li. An efficient insertion operator in dynamic ridesharing services. In *ICDE*, pages 1022–1033, 2019.

[58] Y. Zeng, Y. Tong, L. Chen, and Z. Zhou. Latency-oriented task completion via spatial crowdsourcing. In *ICDE*, pages 317–328, 2018.

[59] B. Zhao, P. Xu, Y. Shi, Y. Tong, Z. Zhou, and Y. Zeng. Preference-aware task assignment in on-demand taxi dispatching: An online stable matching approach. In *AAAI*, pages 2245–2252, 2019.

[60] L. Zheng, P. Cheng, and L. Chen. Auction-based order dispatch and pricing in ridesharing. In *ICDE*, pages 1034–1045, 2019.