TrendSharing: a Framework to Discover and Follow the Trends for Shared Mobility Services

Jiexi Zhan [1], Han Wu [1], Peng Cheng [1], Libin Zheng [2], Lei Chen [3], Chen Jason Zhang [4], Xuemin Lin [5], Wenjie Zhang [6]
[1]*East China Normal University, Shanghai, China*
[2]*Sun Yat-sen University, Guangzhou, China*
[3]*HKUST(GZ) and HKUST, Guangzhou & Hong Kong SAR, China*
[4]*The Hong Kong Polytechnic University, HK SAR, China*
[5]*Shanghai Jiaotong University, Shanghai, China*
[6]*The University of New South Wales, Sydney, Australia*
{jxzhan, han.wu}@stu.ecnu.edu.cn; pcheng@sei.ecnu.edu.cn; zhenglb6@mail.sysu.edu.cn;
leichen@cse.ust.hk; jason-c.zhang@polyu.edu.hk; xuemin.lin@gmail.com; wenjie.zhang@unsw.edu.au

***Abstract*—With the development of ubiquitous smart devices, shared mobility services, such as food delivery, ridesharing and crowdsourced parcel delivery, and the related problems, such as task assignment and route planning have drawn much attention from academia and industry. Specifically, shared mobility services enable one worker to deliver more than one package/passenger together such that their routes can share some common sub-routes. Tardiness (the exceeded time) can harm users' experience and reduce the revenue of workers and platforms, which is not well handled in the existing studies. In this paper, we propose a framework, TrendSharing, to minimize the total tardiness when serving all tasks. In TrendSharing, we first build a flow tree to group tasks together. Then, we propose a concept of trend, which represents a group of tasks with high sharability in the flow tree. Furthermore, we devise a decision factor ϵ-*score* to properly select the trend from the flow tree. In addition, we devise an indicator *k-regret* to quantify the likelihood of tardiness for each task and devise a greedy algorithm to conduct task assignment. We observe that the insertion operation that is widely used by existing works has little effect on the objective of minimizing total tardiness. Thus, we adopt a simple yet effective strategy, which will continuously append newly planned routes to the workers' existing routes. Moreover, we design an algorithm to plan a route for the trend with an approximation ratio of 2.5. Through extensive experiments, we demonstrate the efficiency and effectiveness of our proposed approaches on real datasets.**

I. Introduction

The ubiquitous smart devices and advanced navigation systems collaboratively provide us a large amount of spatial data and consequently spawn a variety of modern applications such as ridesharing (e.g., Uber [1], Didi [2], Lyft [3]), food delivery (e.g., Grubhub [4], Meituan [5], Eleme [6]), package delivery (e.g., UPS [7], Cainiao [8]) and so on. Users are able to submit tasks which specify the origins and destinations to these platforms. The platforms will estimate the expected delivery time (EDT) of each task and return it to the users. Periodically, the platforms will assign the newly arrived tasks to the available workers. The workers need to arrive at the origins to pick up the package/passengers and deliver them to the destinations. Since each worker can accept multiple tasks at the same time, after the task assignment step, the platforms also need to plan routes for the workers so that they can complete the assigned tasks on time.

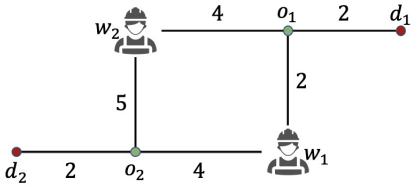


Fig. 1. A motivation example

However, in real life, the number of workers is limited. Thus, it is inevitable that some tasks will be tardy, especially when the weather is poor, the traffic is congest and the demand of users is overwhelming. Nobody likes tardiness, since it will severely affect the user experience and reduce the revenue of workers and platforms. To alleviate the burden of workers while satisfying users' real-time requirements, in this paper, we study the problem of minimizing total tardiness when serving all tasks. To the best of our knowledge, none of existing works considers the tardiness as the optimization objective. Next, we will use the following example to illustrate our motivation.

Example 1 (Motivation Example)**.** *As shown in Figure 1, there are two workers w_1, w_2 and two tasks t_1 and t_2. The origins and destinations are denoted by o_i and d_i respectively ($i = 1, 2$). The numbers on the edges represent the travel distance. Assume that both tasks are submitted to the platform at time 0 and the expected delivery time of them are both 6. And the speed of the two workers are both 1. If we assign t_1 to w_2 and t_2 to w_1, both tasks will be delivered on time, and the total tardiness will be zero. However, if the objective is to minimize total travel time like existing works, we may assign t_1 to w_1 and t_2 to w_2, which will result in a shorter total travel time ($11 < 12$). But in this case, t_2 will be tardy and the total tardiness will be 1, which is worse than the previous assignment. Thus, it is vital to devise algorithms specific to the tardiness.*

This minimization problem is composed of two challenging subproblems: task assignment problem and route planning problem. For the task assignment problem, most existing works model it as a bipartite graph matching problem and regard the maximization of the number of served tasks as the primary objective [9], [10], [11], [12]. Usually, there will be a secondary objective, which could be maximizing the total revenue of the platform [13], [14], [15], [16] or minimizing the total travel cost of the workers [9], [12]. To solve this matching problem, there are mainly two kinds of solutions. One is the maximum flow based solutions, which reduce the problem to the maximum flow problem and solve it by classical algorithms (e.g., Ford-Fulkerson algorithm [17]) [9], [12]. The other one is greedy solutions, which will greedily pick the most valuable task-worker pair according to the objectives [18], [19]. Since all tasks need to be served in our problem, and there is no limit of the workers' working areas, each task may have many candidate workers, which will result in a dense bipartite graph. Using maximum flow algorithms on such a graph is quite time-consuming. Thus, we explore greedy solutions.

Intuitively, tasks in the area with plenty of workers are less likely to time out, but tasks in remote areas have a higher probability of being tardy. Based on this observation, we greedily pick the task that is more easily to be tardy and assign it to the worker who can serve it with minimal tardiness, ties are broken by minimal travel time. We devise an indicator called *k-regret* to quantify the likelihood of tardiness for each task and use it to determine the order of tasks being selected. What's more, tasks with high sharability can be grouped together and assigned to the same worker, which will significantly improve the efficiency of workers and reduce the possibility of tardiness. We leverage a spatial index called *hierarchically well-separated tree (HST)* to capture the spatial characteristics of tasks and propose a novel structure called *flow tree* to group tasks together. Then, we define a concept of trend, which represents a group of tasks with high sharability in the *flow tree*. To select the trend from the *flow tree*, we propose a decision factor *ϵ-score* to guide the procedure.

For the route planning problem, most existing works rely on a key operation called insertion, which can insert the origin and destination of a task into a worker's existing route with minimal increased distance. It was first proposed by Jaw et al. in 1986 to solve the dial-a-ride problem [20]. Recently, it is widely used in large-scale ridesharing problems to incrementally update a partial route, which can avoid the expensive computational cost of reordering and obtain a satisfying result on the objective of minimizing total distance [21], [22], [23], [24], [25], [26], [27]. Although it performs well on the ridesharing problems, it has little effect in our problem through our experiments. The main reason is that our problem has no deadline constraint, namely, we can serve a task even though the actual delivery time exceeds its expected delivery time. Thus, the optimal insertion position found by the operation is likely to postpone the arrival time of the later positions and greatly increase the total tardiness.

To overcome this issue, one straightforward idea is to change the objective of the insertion operation to minimize increased tardiness. However, the time overhead of finding the optimal position will be quite large because we need to enumerate all possible positions ($O(n^2)$), and for each of them, we need to recalculate the arrival time of the positions behind it and calculate the increased tardiness ($O(n)$). Thus, the overall time complexity is $O(n^3)$ (n is the number of positions in the existing route). The technique used to optimize the original insertion operation [25] cannot be applied to the modified version. Therefore, the insertion operation is not suitable for our problem. We observe that simply appending newly planned route into the worker's existing route can achieve a good result in terms of tardiness. The main reason is that append will not delay the tasks assigned earlier. Moreover, since we have selected tasks in a trend as an assignment unit, workers can quickly complete assigned tasks first, then serve new tasks as early as possible. To plan a route for the tasks in a trend, we propose a route planning algorithm with theoretical guarantee. To summarize, we make the following contributions:

- We consider the tardiness as the optimization objective, propose a Minimum Tardiness Task Assignment and Route Planning (MTARP) problem and prove it is not only NP-hard, but also inapproximable in Section III.
- We propose the *flow tree* to group tasks together and devise a decision factor *ϵ-score* to select the *trend* from the *flow tree* as an assignment unit in Section IV. Then, we take each task's likelihood of tardiness into consideration and design a greedy algorithm to conduct task assignment in Section V.
- We take the advantage of trend and design an effective route planning algorithm with theoretical guarantee for the trend in Section VI.
- We have conducted extensive experiments on real datasets to show the efficiency and effectiveness of our proposed solutions in Section VII.

In addition, we review and compare related works in Section II and conclude our work in Section VIII.

II. Related Work

A. Traditional Routing Problems

The study of task assignment and route planning problem can be traced back to the Vehicle Routing Problem (VRP) proposed by George Dantzig and John Ramser in 1959, which needs to find a set of routes for a fleet of vehicles to deliver commodities from a depot to the customers with minimum total travel cost [28]. This is a classical combinatorial optimization and NP-hard problem [29], [30]. Pickup and Delivery Problem (PDP) is an important extension of VRP, which needs to transport commodities between origins and destinations with minimum total travel cost [31], [32]. Dial-a-ride Problem (DARP) is a variant of the one-to-one PDP, which aims to plan a set of minimum cost vehicle routes capable of accommodating as many users as possible [33]. These problems are often modeled as integer programming problems and solved by exact algorithms (e.g., branch and bound [34], branch and

cut [35], etc.) or heuristic algorithms (e.g., large neighborhood search [36], [37], [38], tabu search [39], etc.). These methods are time-consuming, and can only solve small instances.

B. Large-Scale Ridesharing Problems

Most existing works on ridesharing have deadline constraints. A task must be completed before its deadline, otherwise, will be rejected. Ma et al. propose a ridesharing platform called T-share and try to maximize the number of served tasks and minimize total travel distance [21]. They use the grid index to partition the road network and reduce the time cost of the shortest path computation. Based on the grid index, they devise a double-end search algorithm to filer the available workers and use the insertion operation to choose the feasible worker with minimal increased distance to accept the newly arrived task. Thangaraj et al. also propose a ridesharing platform called XAR and devise a hierarchical index to mitigate the shortcoming of grid index and integrate a multi-model trip planner to build an integrated transportation system [24].

Huang et al. propose a data structure called kinetic tree that maintains all possible routes of a worker [23]. When a new task is assigned to the worker, the tree needs to update and prune impossible routes. Kinetic tree can enhance efficiency by avoiding recomputing the possible routes from scratch when inserting a new task. However, it cannot be applied in problems without deadline constraints like ours because the prune step is highly dependent on the deadlines of tasks. Deng et al. model the task assignment problem as a bipartite matching problem and leverage the maximum flow algorithm to find a maximum matching. To accelerate the algorithm, they propose a partitioning-based algorithm to reduce the number of edges in a bipartite graph. They also use the insertion operation to schedule the assigned tasks [12]. Tong et al. propose a unified cost function that can unify the objectives of minimizing total travel distance, maximizing the number of served tasks and total revenue of the platform by one formula [25]. Moreover, they optimize the insertion operation to $O(n)$ time complexity by dynamic programming. Wang et al. take the future demand into consideration and extend the insertion operation with $O(n^2)$ time complexity to balance the demand and supply, which will gain the largest profit for a long period [27]. Zeng et al. aim to maximize the total revenue of the platform. They prove that a simple greedy algorithm that always picks the most valuable group of tasks has an approximation ratio of 0.5. To reduce the time complexity of enumerating all combinations of tasks, they further propose a data structure called additive tree to prune useless combinations [16].

Yuan et. al. survey the recent traffic generator for simulation evaluation, including the ridesharing scenarios [40].

C. Machine Scheduling Problems

Minimizing total tardiness is a commonly used objective in machine scheduling problems [41], [42], [43], [44]. However, there are two major differences between our problem and machine scheduling problems: 1) the processing time of jobs in machine scheduling problems are known in prior and will not be affected by the order being processed, but the processing time of tasks in our problem is not fixed, which is jointly determined by tasks' and workers' real-time locations; 2) the machines are static and homogeneous, but the workers will move around at different speeds according to previously planned routes.

TABLE I
SUMMARY OF NOTATIONS USED IN OUR WORK.

Notation	Meaning
W	A set of m workers
l_i	Current location of worker w_i
cap_i	Capacity of worker w_i
spd_i	Speed of worker w_i
S_i	Current route of worker w_i
o_j, d_j	Origin and destination of task t_j
r_j, ed_j, ad_j	Release time, expected and actual delivery times of t_j
wei_j	Weight of task t_j
ω_j	*k-regret* of task t_j, defined in equation 2
Δt	Batch interval

III. PROBLEM DEFINITION

The Minimum Tardiness Task Assignment and Route Planning (MTARP) problem studied in this paper is a dynamic problem, namely, tasks will continuously arrive as the time goes by. Generally, there are two modes for dynamic problems [45]. One is the online mode, which will assign a worker to a task immediately once the task has arrived; the other one is the batch-based mode, which will assign tasks at regular intervals. We adopt the batch-based mode, because it is more suitable for the real scenarios and more general (the online mode can be regarded as a special case of the batch-based mode, that is, there is at most one task in each batch). We first define notations related to the MTARP problem.

A. Preliminaries

Definition 1 (Workers)**.** Let $W = \{w_1, w_2, \ldots, w_n\}$ denote a set of n workers. Each worker $w_i \in W$ is initially at location l_i and maintains his/her current route S_i that composed of a series of locations to be visited. Moreover, each worker w_i has his/her own capacity cap_i, which is the maximum sum of tasks' weight that he/she can carry at the same time, and travels with a speed of spd_i m/s.

Definition 2 (Tasks)**.** Let $T = \{t_1, t_2, \ldots, t_m\}$ denote a set of m tasks. Each task $t_j \in T$ is released at timestamp r_j from the requester with the origin o_j, the destination d_j, a weight wei_j and the expected delivery time ed_j. Let the actual delivery time of task t_j be ad_j, then the tardiness is calculated as:

$$tardiness = \max(0, ad_j - ed_j)$$

Definition 3 (Route)**.** A route $S_i = [l_i^0, l_i^1, \ldots, l_i^k]$ of a worker w_i consists of a series of temporally-ordered locations. $l_i^0 = l_i$ is the initial location of w_i. l_i^1 to l_i^k are the origins and destinations of assigned tasks. A route S_i is valid if and only if:

- (capacity constraint) The total weight of tasks in S_i is no more than the capacity of w_i at any time;

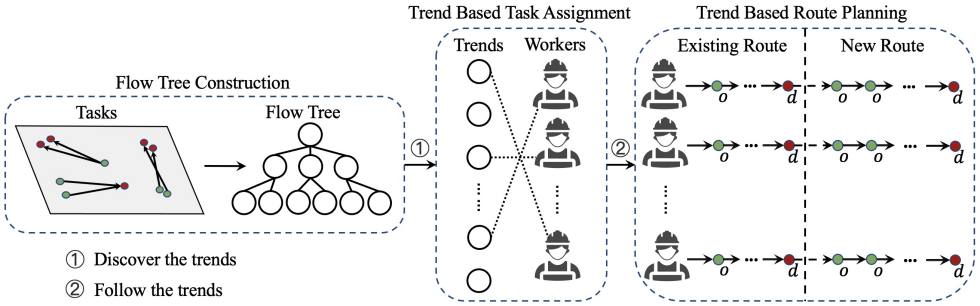


Fig. 2. An overview of the TrendSharing framework

- (order constraint) For any task t_j, its origin o_j appears before destination d_j in S_i.

Definition 4 (Batch)**.** Let B_i denote the ith batch with an interval Δt. Suppose B_i starts at timestamp t^i, then, it will end at timestamp $t^i + \Delta t$. The tasks whose release time is within $[t^i, t^i + \Delta t)$ will be assigned in the ith batch.

Definition 5 (Metric Space)**.** A metric space M is composed of a set of points V and a function d. $d(x, y)$ represents the distance of a pair of points $x, y \in V$ and satisfies the following three properties:

- (identity) $d(x, y) \geq 0$ and $d(x, y) = 0 \Leftrightarrow x = y$;
- (symmetry) $d(x, y) = d(y, x)$;
- (triangle inequality) $d(x, y) + d(y, z) \leq d(x, z)$.

Definition 6 (Minimum Tardiness Task Assignment and Route Planning (MTARP) Problem)**.** Given a set of workers W, a set of tasks T and a batch interval Δt (the system conducts a task assignment every Δt seconds), MTARP problem aims to plan a route S_i for each worker $w_i \in W$ such that the total tardiness of all tasks is minimized:

$$min \sum_{t_j \in T} max(0, ad_j - ed_j)$$

and to satisfy the following constraints:

- Tasks have been assigned cannot be reassigned to a different worker;
- The planned routes must be valid, namely, they cannot violate the capacity and order constraints.

B. Hardness Analysis

Theorem III.1. *The MTARP problem is NP-hard and inapproximable.*

Proof. We will prove the above theorem through a reduction from the minimizing average flow time for online dial-a-ride problem (F_{avg}-$OLDARP$) [46], which is NP-hard and inapproximable.

Given a metric space $M =< V, d >$, an instance $I_{F_{avg}-OLDARP}$ of F_{avg}-$OLDARP$ in M consists of a single server s and a set of m requests $R = \{r_1, r_2, \ldots, r_m\}$. Each reqeust is a triple $r_i =< t_i, o_i, d_i >$, where t_i is the release time, o_i is the origin and d_i is the destination. It is assumed that $\{r_1, r_2, \ldots, r_m\}$ is given in order of non-decreasing release times, namely $0 \leq t_1 \leq t_2 \leq \ldots \leq t_m$. The server s is located at origin $o \in V$ at time 0 and can move at constant unit speed spd. It will fulfill all requests, and it can carry at most C objects at a time. Let the completion time of request r_i be C_i, the objective of $I_{F_{avg}-OLDARP}$ is to minimize the average flow time, i.e. $\min \frac{1}{m} \sum_{i=1}^{m} C_i - t_i$.

From above definition, we can construct an instance of MTARP problem I_{MTARP} from $I_{F_{avg}-OLDARP}$ by the following steps: 1) create a single worker w who is initially at position o at time 0 and the speed and capacity is equal to spd and C respectively; 2) create a set of tasks T, $|T|$=$|R|$ and for each $t_i \in T$, $r_i = t_i, ed_i = t_i$; 3) use the same metric space M. By setting the expected delivery time equal to corresponding release time of all tasks, the objective of I_{MTARP} is essentially the same as $I_{F_{avg}-OLDARP}$ because we just need to divide the result of I_{MTARP} by the number of tasks m, then we can get the result of $I_{F_{avg}-OLDARP}$. Since F_{avg}-$OLDARP$ is reducible to MTARP problem that is NP-hard and inapproximable, we can conclude that MTARP problem is also NP-hard and inapproximable. □

C. An Overview of the TrendSharing Framework

As shown in Figure 2, the framework consists of three components: flow tree construction, trend based task assignment and trend based route planning.

We first leverage a spatial index HST to cluster the origins and destinations of tasks together. Then, we build a flow tree in accordance with the HST to group tasks together. Each node in the tree contains a set of tasks whose origins and destinations are within the same circular region, respectively. To properly select the node in the flow tree, we propose a concept of trend and devise a decision factor *ϵ-score* to discover the trends.

Then, we devise an indicator *k-regret* to quantify the likelihood of tardiness for each task and determine the order of tasks being selected. For each selected task, we find the worker who can serve it with minimal tardiness, then select the trend that contains the select task and assign all tasks in the trend to the worker.

Finally, benefit from the property of trend, we design an effective route planning algorithm for tasks within a trend with an approximation ratio of 2.5 to the optimum. Since the insertion operation with the objective of minimizing detour is likely to postpone the delivery time of the existing tasks and greatly increase the total tardiness, and the insertion operation with the objective of minimizing increased tardiness is too time-consuming. We adopt the strategy of appending the newly planned route to the existing route, which is quite efficient. Our experiment demonstrates that this strategy is quite effective with regard to the objective of minimizing total tardiness.

Algorithm 1: HST Construction Procedure

Input: A metric space $M = (V, d)$
Output: A HST $\mathcal{T}$

1 sample a permutation π of V and β from $[0.5, 1)$
2 $\Delta \leftarrow max_{v_i, v_j \in V} d(v_i, v_j), H \leftarrow \lceil log_2 \Delta \rceil$
3 $NS_H \leftarrow \{\{V\}\}$
4 **for** $i \leftarrow H - 1$ *to* 0 **do**
5 $r_i = 2^i \cdot \beta, NS_i \leftarrow \emptyset$
6 **foreach** *node* $N \in NS_{i+1}$ **do**
7 **for** $j \leftarrow 0$ *to* $n - 1$ **do**
8 $N' \leftarrow \{v \in N \mid d(\pi[j], v) < r_i\}$
9 add N' to NS_i and remove $v \in N'$ from N
10 **return** $\mathcal{T} \leftarrow NS_H, NS_{H-1}, \ldots, NS_0$

In the batch-based mode, we consider the tasks released within this batch as a set, and build an HST and flow tree for the tasks in this set, ensuring that tasks fulfilling the trend similarity requirement are categorized together. Then, in the order of *k-regret* from large to small, we allocate the most suitable worker and plan the route for a task or task group. We update the worker's current location at the end of each batch.

IV. Discover the Trend

Tasks with high sharability can be grouped together and assigned to the same worker, which will significantly improve the efficiency of workers and reduce the possibility of tardiness. To make full use of the spatial features of tasks, in this section, we formally model the sharability of tasks. Specifically, we first briefly introduce a spatial index called HST to capture the spatial characteristics of tasks in Section IV-A. Then, we propose a new tree structure termed flow tree to group tasks together in Section IV-B. Finally, we propose a concept of trend to model the sharability of tasks and devise a decision factor ϵ-*score* to properly select the trend from the flow tree in Section IV-C.

A. Spatial Index HST

HST was proposed by Bartal in 1996 [47]. It can embed any metric space into a tree space, which guarantees that the distance stretch is tightly bounded by $O(\log n)$ [48]. This property is used by many works to design approximation algorithms [48], [26], [49], [50]. However, since our problem is inapproximable, we mainly use it to capture the spatial characteristics of the tasks and better guide the task assignment. Next, we introduce the HST construction procedure.

As shown in Algorithm 1, given a metric space $M = (V, d)$, we first choose a random permutation π of the points in V and a parameter β uniformly from $[0.5, 1]$ in line 1. Then we calculate the diameter of V to determine the height of the tree in line 2. After that, we initialize the root of HST, which contains all points in V in line 3. Nodes in the same level are disjoint and the union of them includes all points in V. From lines 4 to 12, we iteratively build the tree through a top-down style. In the ith level, we first determine the radius

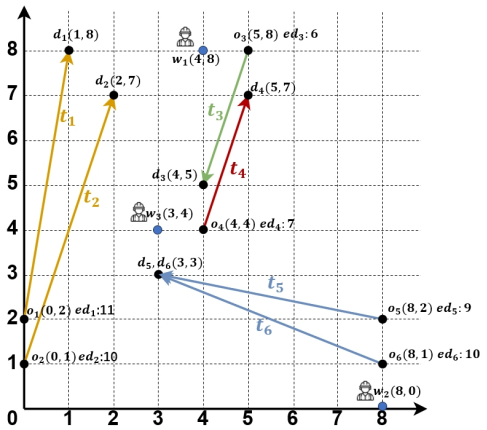


Fig. 3. A toy instance of our problem

r_i, and initialize the node set NS_i as an empty set in line 5. Then, we traverse each node N in the previous level and try to partition it with a smaller circle in lines 6-11. Finally, we obtain all the nodes in HST and complete the construction.

Example 2 (An Example of HST)**.** *We use an example to explain the construction procedure of HST. Suppose we have 6 tasks in a Euclidean space, as shown in Figure 3. And the origins and destinations of these tasks form the point set V. Let the permutation of points be $\pi = \{o_1, d_1, o_2, d_2, \ldots, o_6, d_6\}$ and $\beta = 0.5$, we calculate the diameter of the point set, which is $d(d_1, o_6) = \sqrt{98} \approx 9.90$. Thus, the height of HST is $\lceil \log_2 \Delta \rceil = 4$. The root node of HST contains all the points. In level 3, the radius of the circular region is $2^3 \cdot 0.5 = 4$, and we try to partition the root with the decreased circle. We traverse the permutation π to choose the center point. Firstly, we choose o_1 as the center point and cluster it with o_2. Secondly, we choose d_1 and cluster it with d_2. Thirdly, we choose o_2 and there is no new node constructed. When we traverse to d_2, although it has been included in another node, it can still be a center point and form a new node which contains o_3, o_4, d_3 and d_4. We repeat above process until all points have been partitioned into new nodes of HST. The processes in other levels are the same. The constructed HST is shown in Figure 4.*

B. Flow Tree for Shared Mobility

After HST is constructed, the origins and destinations of tasks are clustered into different nodes in HST. Each node can be regarded as a circular region. Through adding directed edges between nodes at the same level, we can clearly observe how tasks flow from one region to another region. Based on this observation, we build a new tree structure called flow tree in accordance with the added edges. Each node in the tree contains a group of tasks, whose origins and destinations are within the same circular regions, respectively. From Algorithm 1, we know that the granularity of clustering of HST is finer and finer from top to bottom because the radius of the circle is getting smaller and smaller. Flow tree also preserves this property. Next, we will introduce the construction procedure of the flow tree.

Algorithm 2: Flow Tree Construction Procedure

Input: A set of tasks T, a HST $\mathcal{T}$
Output: A flow tree $\mathcal{T}^f$

1 $FNS_H \leftarrow \{\{T\}\}$
2 **for** $i \leftarrow \mathcal{T}.H - 1$ *to* 0 **do**
3 $map \leftarrow \{(<,>,\{\})\}$
4 **foreach** $t_j \in T$ **do**
5 $id_p \leftarrow$ ID of the node containing p_j in ith level
6 $id_d \leftarrow$ ID of the node containing d_j in ith level
7 add t_j to map with key $< id_p, id_d >$
8 **foreach** $< id_p, id_d > \in map.keys$ **do**
9 add all tasks with key $< id_p, id_d >$ to TDS_i
10 **return** $\{S_i \mid w_i \in W\}$

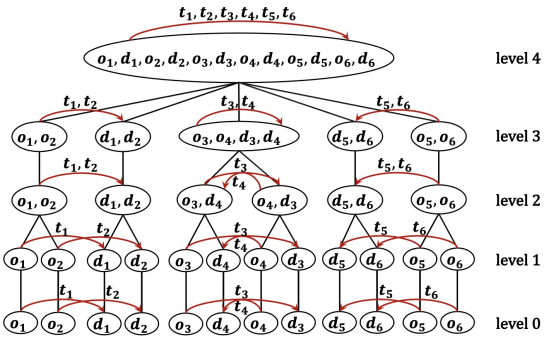


Fig. 4. HST constructed according to the origins and destinations of tasks in the toy example

Algorithm 2 shows the construction procedure of the flow tree. Nodes in the same level are disjoint and the union of them includes all tasks in T. From lines 2 to 12, we iteratively build the tree in accordance with the information provided by HST. In each level, we first initialize a map whose key is a pair of node IDs in HST, and the value is a set of tasks in line 3. Then we group the tasks together by their corresponding keys in lines 4-8 and construct the nodes based on the grouped tasks in lines 9-11.

Example 3 (An Example of Flow Tree)**.** *In Figure 4, each red edge contains a number of tasks whose origins and destinations are within the same nodes respectively in HST. And we can build a flow tree as shown in Figure 5 according to these edges. The root of the tree contains all tasks and each leaf node contains only one task. Here we take level 3 for example to show the construction procedure. Suppose the node ids in level 3 of HST are* $[1, 2, 3, 4, 5]$ *from left to right. Then the keys for* t_1 *to* t_6 *are* $< 1, 2 >, < 1, 2 >, < 3, 3 >, < 3, 3 >, < 5, 4 >, < 5, 4 >$*. By grouping the tasks by their keys, we can obtain the final groups as* $\{t_1, t_2\}, \{t_3, t_4\}$ *and* $\{t_5, t_6\}$.

C. Trend

In real world, we can observe that tasks will present a certain trend. For example, in food delivery scenario, tasks often show a tendency to scatter from the center to the periphery because the trading area is relatively concentrated; in ridesharing scenario, tasks often flow from suburb to downtown during the morning peak and reverse during the evening peak. Certainly,

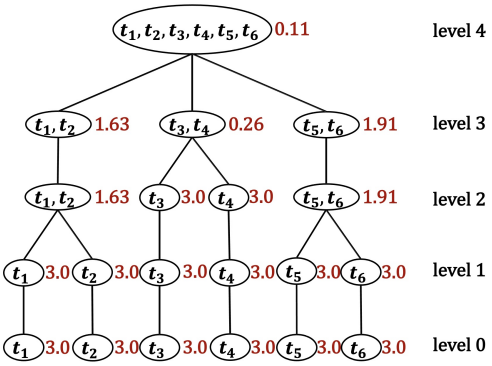


Fig. 5. Flow tree constructed according to the added edges of HST in Figure 4

tasks with the same trend have high sharability, and we hope to assign them to the same worker so that he/she can efficiently deliver all of them, which can significantly reduce the detour and possibility of tardiness, especially in the rush hour. Thus, we propose a concept of trend, which is defined as follows.

Definition 7 (Trend)**.** A trend is a group of tasks that can be assigned to a single worker such that the form of the optimal route to deliver them is first traversing all origins to pick up the goods/passengers, then delivering them one by one.

Not all nodes in the flow tree can meet the requirement of trend. For example, in Figure 3, t_3 and t_4 are in completely opposite directions, but they are within the same node of the flow tree in level 3 as shown in Figure 5. Thus, we devise a decision factor *ϵ-score* to help us select the trend from the flow tree. The formula to calculate it is shown in equation 1.

$$\epsilon\text{-}score = \frac{3 \min d_{inter}}{2 \max_{d_{inner}} + max_{d_{inter}}} \tag{1}$$

In equation 1, d_{inter} is the distance of an *inter-edge* that connects an origin and a destination, d_{inner} is the distance of an *inner-edge* that connects origins or destinations. The intuition behind this formula is that if replacing an *inner-edge* with an *inter-edge* has no gain at all, we can conclude that the optimal form of the route to deliver all tasks is first traversing all pickup locations, then delivering them all in one shot. Next, we propose a lemma to formally prove it.

Lemma IV.1. *For each node in flow tree, if the corresponding* ϵ*-score* > 1.0*, it satisfies the requirement of a trend.*

Proof. When a node contains a single task, $d_{inner} = 0$ and $\min d_{inter} = \max d_{inter}$ because there is only one origin and one destination.Thus, its *ϵ-score* is equal to 3.0. Obviously, it satisfies the requirement of a trend. Otherwise, suppose the number of tasks is n, then there will be $2n - 1$ edges in the route. Let R_{OD} denote the optimal route that traversing origins first then destinations and R_{OTR} denote the optimal route with other forms. R_{OD} is composed of $2n - 2$ *inner-edges* and 1 *inter-edge*, suppose R_{OTR} replaces $2k$ *inner-edges* by *inter-edges*, as a result, it is constitued by $2n - 2 - 2k$ *inner-edges* and $1 + 2k$ *inter-edges* $(1 \leq k < n)$. If $\min d_{inter} > \max d_{inner}$, obviously, substituting $2k$ *inner-edges* with $2k$ *inter-edges* has no gain at all when the original *inter-edge*

Algorithm 3: Trend Based Task Assignment

Input: A set of tasks T, a set of workers W
Output: A route S_i for each worker $w_i \in W$

1 build flow tree $\mathcal{T}^f$ according to T
2 sort $t_j \in T$ by ω_j in descending order
3 **foreach** $t_j \in T$ **do**
4 **if** t_j *has been assigned* **then**
5 **continue**
6 find worker w_i with the minimal tardiness for t_j
7 $td \leftarrow$ largest trend w_i can accommodate in $\mathcal{T}^f$ that contains t_j
8 update route S_i for w_i with algorithm 4
9 **foreach** $t_j \in T$ **do**
10 remove t' from td
11 recursively remove t' from td's children
12 recursively remove t' from td's parent
13 **return** $\{S_i \mid w_i \in W\}$

is preserved. Otherwise, assume the extreme condition that original *inter-edge* is the longest, and it is replaced by the shortest *inter-edge*, then the gain is $\max d_{inter} - \min d_{inter}$. The minimum loss of the other $2k$ substitutions is achieved when $k = 1$, namely, $2(\min d_{inter} - \max d_{inner})$. Thus, when $\max d_{inter} - \min d_{inter} - 2(\min d_{inter} - \max d_{inner}) < 0$, the travel time of R_{OD} is less than that of R_{OTR}. This is equivalent to ϵ-$score > 1$, which completes our proof. □

Example 4 (An Example of ϵ-*score*)**.** *In Figure 5, the number to the right of the node is its ϵ-score. Take the first node in level 3 as an example,* $\max d_{inter} = d(p_2, d_1) = \sqrt{50}, \min d_{inter} = d(p_1, d_2) = \sqrt{29}, \max d_{inner} = d(d_1, d_2) = \sqrt{2}$*, thus,* $\epsilon\text{-}score = \frac{3\sqrt{29}}{2\sqrt{2}+\sqrt{50}} \approx 1.36$*. Thus, this node is a trend. This is also true for the last node in level 3 because its ϵ-score is also > 1.0. However, the middle node in level 3 is not a trend because its ϵ-score is negative. We can also validate these assertions from Figure 3 where tasks within the same trend are marked by the same color.*

V. Trend Based Task Assignment

A. Basic Idea

The main reason for tardiness is the lack of workers. If the origin of a task is located in an area with abundant workers, there are many choices for it to be delivered on time. In contrast, tasks in remote areas have fewer choices, and they should be considered with higher priority. Based on this intuition, we propose an indicator called *k-regret* to quantify the likelihood of tardiness for each task and use it to determine the order of tasks being selected.

Let x_{jk} indicate the worker for which task t_j has the k'th minimal tardiness, and $\delta_{j,i}$ denote the tardiness of the task t_j when assigned to worker w_i. Thus, we can calculate the *k-regret* ω_j for task t_j as follows:

$$\omega_j = \sum_{i=1}^{k} \delta_{j,x_{ji}} - \delta_{j,x_{j1}} \tag{2}$$

The *k-regret* of a task is the sum of the difference in the tardiness of assigning it to its best worker and its k'th best worker. A task with larger *k-regret* means considering it later will incur worse results.

TABLE II
Tardiness of tasks.

t_1	$(w_3, 0.00)$	$(w_1, 2.29)$	$(w_2, 3.33)$
t_2	$(w_3, 0.57)$	$(w_1, 4.39)$	$(w_2, 4.39)$
t_3	$(w_1, 0.00)$	$(w_3, 1.63)$	$(w_2, 5.71)$
t_4	$(w_3, 0.00)$	$(w_1, 0.16)$	$(w_2, 1.82)$
t_5	$(w_2, 0.00)$	$(w_3, 1.48)$	$(w_1, 3.31)$
t_6	$(w_2, 0.00)$	$(w_3, 1.22)$	$(w_1, 3.45)$

Example 5 (An Example of *k-regret*)**.** *As shown in Figure 3, there are three workers located in different areas. The tardiness of tasks when assigned to different workers is shown in table II and each row is sorted by the tardiness in ascending order. Suppose $k = 3$, then we can calculate the k-regret of tasks and sort the results by descending order:* $[(t_2, 7.64), (t_3, 7.34), (t_1, 5.62), (t_5, 4.79), (t_6, 4.67), (t_4, 1.98)]$*. We can observe that t_2 has the largest k-regret. The reason is: for t_2, w_3 is the nearest worker and can deliver it with tardiness 0.57. If w_3 is not assigned to it, then there are just two choices left for it, which will make t_2 waiting for a long time to be completed and significantly increase the tardiness. Thus, we should consider t_2 first and assign w_3 to it. We can also observe that t_4 has the smallest k-regret because there are many workers around its origin and all of them can complete it with small tardiness. Thus, considering it at last will not impact the result too much.*

K-regret can help us capture the temporal features of tasks. On the other hand, spatial features are also important for the objective of minimum tardiness. We take the advantage of the flow tree to find a trend that contains the selected task and assign them as a whole to a specific worker such that the time to deliver all of them will not exceed the time delivering any single of them too much. Based on these considerations, we propose a trend based task assignment.

B. Algorithm Detail

Algorithm 3 presents our task assignment algorithm. In line 1, we first build a flow tree based on the tasks in the current batch. In line 2, we sort the tasks by their *k-regret* in descending order. Then in lines 3-7, we iterate each task t_j, if t_j has not been assigned, we find the worker w_i who can deliver t_j with minimal tardiness to serve it. Then, we start from the leaf node that contains t_j, go upward to find the largest trend td that w_i can accommodate in the flow tree in line 8. In line 9, we invoke the route planning algorithm to update w_i's existing route according to tasks in the trend td. In lines 10-14, we remove the assigned tasks from the flow tree. Next, we will use an example to illustrate our task assignment procedure.

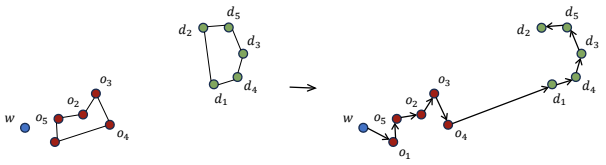


Fig. 6. Construct a route by merging two cycles.

Algorithm 4: Trend Based Route Planning

Input: A set of tasks T, one worker w_i with route S_i
Output: An updated route S_i for worker w_i

1 $G_o = (V_o, E_o), V_o = \{o_j \mid t_j \in T\}, E_o = V_o \times V_o$
2 $G_d = (V_d, E_d), V_d = \{d_j \mid t_j \in T\}, E_d = V_d \times V_d$
3 $cycle_o \leftarrow Christofides(G_o)$
4 $cycle_d \leftarrow Christofides(G_d)$
5 $cost \leftarrow INF$, $(s_o, t_o, s_d, t_d) \leftarrow (\ ,\ ,\ ,\ )$
6 **foreach** *edge* $e_o \in cycle_o$ **do**
7 $o_1, o_2 \leftarrow$ end points of e_o
8 **foreach** *edge* $e_d \in cycle_d$ **do**
9 $d_1, d_2 \leftarrow$ end points of e_d
10 $min_{o_1} = \min(d(o_1, d_1), d(o_1, d_2))$
11 $min_{o_2} = \min(d(o_2, d_1), d(o_2, d_2))$
12 **if** $d(w, o_1) + min_{o_2} > d(w, o_2) + min_{o_1}$ **then**
13 $\Delta = d(w, o_2) + min_{o_1} - wei(e_o) - wei(e_d)$
14 **if** $\Delta < cost$ **then**
15 $cost = \Delta$
16 **if** $d(o_1, d_1) < d(o_1, d_2)$ **then**
17 $(s_o, t_o, s_d, t_d) \leftarrow (o_2, o_1, d_1, d_2)$
18 **else**
19 $(s_o, t_o, s_d, t_d) \leftarrow (o_2, o_1, d_2, d_1)$
20 **else**
21 $\Delta = d(w, o_1) + min_{o_2} - wei(e_o) - wei(e_d)$
22 **if** $\Delta < cost$ **then**
23 $cost = \Delta$
24 **if** $d(o_2, d_1) < d(o_2, d_2)$ **then**
25 $(s_o, t_o, s_d, t_d) \leftarrow (o_1, o_2, d_1, d_2)$
26 **else**
27 $(s_o, t_o, s_d, t_d) \leftarrow (o_1, o_2, d_2, d_1)$
28 append $\{s_o, \ldots, t_o, s_d, \ldots, t_d\}$ to S_i
29 **return** S_i

Example 6 (A Running Example of the Task Assignment Procedure)**.** *Continue from the example in Figure 3. We first build a flow tree as shown in Figure 5. Then, we sort the tasks by their k-regret as shown in Example 5. We first select* t_2 *and assign it to* w_3 *because* w_3 *can complete it with minimal tardiness (see Table II). Then, we find the largest trend that contains* t_2 *in flow tree, which is the first node in level 3. After that, we invoke the route planning algorithm to update existing route of* w_3 *as* $[l_3, o_2, o_1, d_2, d_1]$*. The arrival time of* d_1 *and* d_2 *is* 11.63 *and* 10.63 *respectively. Thus, the tardinesses of* t_1 *and* t_2 *are both* 0.63*. Secondly, we will select* t_3 *and assign it to* w_1*. We find that the largest trend in flow tree only contains it, so* w_1*'s route becomes* $[l_1, o_3, d_3]$ *and the tardiness is* 0*. Thirdly, we see* t_1 *has been assigned with* t_2*, so we come to the next one, which is* t_5*. We assign* t_5 *to* w_2 *and find the largest trend in flow tree, which contains* t_5 *and* t_6*. So we update the route of* w_2 *as* $[l_2, o_6, o_5, d_5, d_6]$*. The arrival time of* d_5/d_6 *is* $1+1+\sqrt{26} \approx 7.1$*. Thus, the tardinesses of* t_5 *and* t_6 *are both* 0*. Finally, we will select* t_4*. Note that all three workers' existing routes have been updated, so the worker who can serve* t_4 *with minimal tardiness is not* w_3 *anymore. Instead,* w_1 *will serve* t_4 *with minimal tardiness* 1.32*. Thus, the total tardiness is about* 2.58*.*

C. Complexity Analysis

Let H, m and n denote the height of flow tree, the number of tasks and the number of workers respectively. We need $O(m^2H)$ time to build a flow tree in line 1. Then, we need to calculate the *k-regret* for each task, which needs $O(mn)$ time. The time complexity of sorting in line 2 is $O(m \log m)$. We need $O(n)$ time to find the best worker in line 7, $O(H)$ time to find the largest trend that contains the selected task in line 8 and $O(H)$ time to remove the assigned tasks from the flow tree in lines 10-14. Suppose the time complexity of the route planning algorithm is $O(K)$, then the time complexity in lines 3-15 is $O(m(n+H+K))$. Thus, the overall time complexity is $\max(O(m^2H), O(m(n+H+K)))$. The space cost is mainly from the construction of flow tree. Thus, the space complexity is $O(m^2H)$.

VI. Trend Based Route Planning

A. Basic Idea

Since we adopt the strategy to append a newly planned route to the existing route, the only question left is how to plan a route for a trend. According to Definition 7, we know that the optimal form of a trend's route is first traversing all origins to pick up the goods/passengers, then delivering them to the destinations one by one. Based on this property, we can divide the route planning process into two parts. In either part, we need to plan a route that visits each location exactly once, which is the same as the traditional Hamiltonian path problem. Since the problem is NP-hard [51] and the route planning procedure is required to respond in real-time, we devise an approximation algorithm to efficiently and effectively solve it. Next, we will describe it in detail.

B. Algorithm Details

Algorithm 4 shows the pseudocode of our route planning procedure. Firstly, we construct two complete graphs G_o and G_d according to the origins and destinations in lines 1-2. Then we invoke Christofides algorithm [52] to find Hamiltonian cycles in G_o and G_d in lines 3-4. After obtaining the Hamiltonian cycles, we try to find the best way to link the worker, origins and destinations together. As shown in Figure 6, by removing two edges from the two cycles and adding two edges among the three components, we can construct a complete route that starts from the last location of the worker's existing route and ends in a destination. Specifically, in lines 5, we initialize a variable $cost$ to record current minimal increased travel time and a quadruple (s_o, t_o, s_d, t_d), where s_o, t_o represents the first and last locations of origins and s_d, t_d represents the first and last locations of destinations. In lines 6-27, we enumerate all possible combinations of the two edges which will be removed from the cycles. For each combination, there are four possible routes, which are shown in lines 17, 19, 25 and 27, respectively. We choose the best one among the four possible routes and compare it with the minimum $cost$ we found so far and update the quadruple accordingly. Finally, we are able to construct the complete route by the quadruple and append it to the worker's existing route in line 28.

TABLE III
PARAMETER SETTINGS (NYC).

Parameters	Values
number of tasks $\|T\|$	200K, 250K, **300K**, 350K, 424635
number of workers $\|W\|$	1000, 2000, **3000**, 4000, 5000
batch size Δt (s)	5, 10, **15**, 20, 25
edt ratio μ	1.5, 1.8, **2.0**, 2.3, 2.5

C. Complexity Analysis

Let m denote the number of tasks in a trend. In lines 1-2, we construct two complete graphs, which can be done in $O(m^2)$ time and consumes $O(m^2)$ space. Then we invoke Christofides algorithm which can find an approximate solution in $O(m^2 \log m)$ time. In lines 6-27, there are two nested for loops, and all the operations in the inner loop can be done in $O(1)$ time. Thus, the overall time complexity is $O(m^2)$. In line 36, the final result can be constructed in $O(m)$ time. Finally, the time and space complexity of Algorithm 4 are $O(m^2 \log m)$ and $O(m^2)$ respectively. Note that m is bounded by the maximum capacity of workers, it usually will not exceed 20 in real applications. As a result, the time and space complexity of our algorithm is low enough to support real-time response.

D. Approximation Analysis

Christofides algorithm guarantees finding a solution within 1.5 approximation ratio to the optimum [52]. Benefitting from it, we prove that our algorithm can plan a route with an approximation ratio 2.5 to the optimum for the tasks in a trend and the result is presented in Theorem VI.1.

Theorem VI.1. *Suppose OPT is the optimal route and ALG is the route obtained by our algorithm. Let d_{OPT} and d_{ALG} be the travel time of OPT and ALG respectively. Then we have $d_{ALG} < 2.5 \cdot d_{OPT}$.*

Proof. Suppose the route of OPT is $[w, s_o, \ldots, t_o, s_d, \ldots, t_d]$ and the route of ALG is $[w, s'_o, \ldots, t'_o, s'_d, \ldots, t'_d]$ Thus, the travel time of OPT and ALG are

$$d_{OPT} = d(w, s_o) + d(s_o, \ldots, t_o) + d(t_o, s_d) + d(s_d, \ldots, t_d) \quad (3)$$
$$d_{ALG} = d(w, s'_o) + d(s'_o, \ldots, t'_o) + d(t'_o, s'_d) + d(s'_d, \ldots, t'_d) \quad (4)$$

We denote the optimal Hamiltonian cycles in origins and destinations by TSP^o_{OPT} and TSP^d_{OPT}, and cycles found by our algorithm are TSP^o_{ALG} and TSP^d_{ALG}. Then we have

$$TSP^o_{ALG} \leq 1.5 \cdot TSP^o_{OPT}$$
$$TSP^d_{ALG} \leq 1.5 \cdot TSP^d_{OPT}$$

Since $d(s_o, \ldots, t_o) + d(s_o, t_o) \geq TSP^o_{OPT}$ and $d(s_d, \ldots, t_d) + d(s_d, t_d) \geq TSP^d_{OPT}$, we put them into Equation 3 and can obtain the following inequation

$$d_{OPT} \geq d(w, s_o) + TSP^o_{OPT} - d(s_o, t_o) + d(t_o, s_d) + TSP^d_{OPT} - d(s_d, t_d)$$

Since $d(s'_o, \ldots, t'_o) + d(s'_o, t'_o) = TSP^o_{ALG} \leq 1.5 \cdot TSP^o_{OPT}$ and $d(s'_d, \ldots, t'_d) + d(s'_d, t'_d) = TSP^d_{ALG} \leq 1.5 \cdot TSP^d_{OPT}$, we put them into Equation 4 and can obtain the following inequation:

$$d_{ALG} \leq d(w, s'_o) + 1.5 \cdot TSP^o_{OPT} - d(s'_o, t'_o) + d(t'_o, s'_d) + 1.5 \cdot TSP^d_{OPT} - d(s'_d, t'_d)$$

TABLE IV
PARAMETER SETTINGS (CAINIAO).

Parameters	Values
number of workers $\|W\|$	1000, 1500, **2000**, 2500, 3000
batch size Δt (s)	5, 10, **15**, 20, 25

We denote $\Delta_1 = d(w, s'_o) - d(s'_o, t'_o) + d(t'_o, s'_d) - d(s'_d, t'_d)$ and $\Delta_2 = d(w, s_o) - d(s_o, t_o) + d(t_o, s_d) - d(s_d, t_d)$. Thus, $d_{ALG} \leq \Delta_1 + 1.5 \cdot (TSP^o_{OPT} + TSP^d_{OPT})$ and $d_{OPT} \geq \Delta_2 + (TSP^o_{OPT} + TSP^d_{OPT})$. Then, we have

$$d_{ALG} - d_{OPT} \leq \Delta_1 - \Delta_2 + 0.5 \cdot (TSP^o_{OPT} + TSP^d_{OPT})$$

Next, we will prove that $\Delta_1 - \Delta_2 < TSP^o_{OPT} + TSP^d_{OPT}$:

$$\begin{aligned} \Delta_1 - \Delta_2 = {} & d(w, s'_o) - d(w, s_o) + d(s_o, t_o) - d(s'_o, t'_o) \\ & + d(t'_o, s'_d) - d(t_o, s_d) + d(s_d, t_d) - d(s'_d, t'_d) \end{aligned} \quad (5)$$

According to triangle inequality, we have

$$d(w, s'_o) - d(w, s_o) \leq d(s'_o, s_o)$$
$$d(t'_o, s'_d) - d(t_o, s_d) \leq d(t'_o, t_o) + d(s'_d, s_d)$$

We put them into Equation 5 and can obtain

$$\begin{aligned} & \Delta_1 - \Delta_2 \\ & \leq d(s'_o, s_o) + d(s_o, t_o) + d(t'_o, t_o) - d(s'_o, t'_o) \\ & + d(s'_d, s_d) + d(s_d, t_d) - d(s'_d, t'_d) \\ & < TSP^o_{OPT} + TSP^d_{OPT} \end{aligned}$$

Then, $$\begin{aligned} \frac{d_{ALG}}{d_{OPT}} &= \frac{d_{ALG} - d_{OPT}}{d_{OPT}} + 1 \\ &< \frac{1.5 \cdot (TSP^o_{OPT} + TSP^d_{OPT})}{(TSP^o_{OPT} + TSP^d_{OPT})} + 1 = 2.5 \end{aligned}$$

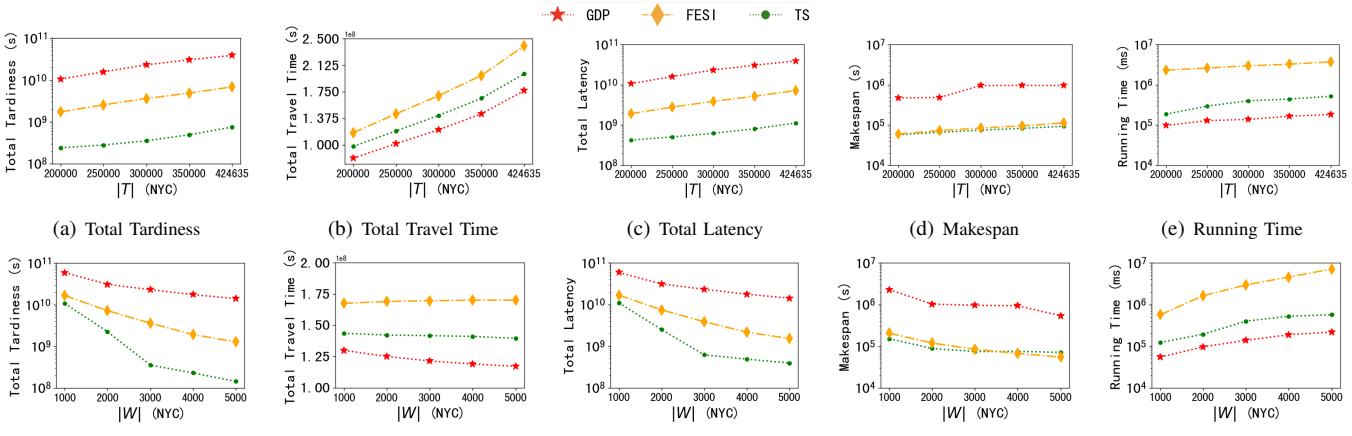
□

VII. EXPERIMENTAL STUDY

In this section, we will present the experimental settings and results of our approaches. Specifically, we will introduce the real datasets we used in Section VII-A and illustrate the compared algorithms and metrics in Section VII-B. Then we will analyze the experimental results of different algorithms in Section VII-C and Section VII-D and summarize the results in Section VII-E.

A. Datasets

Real Datasets. We use two real datasets, i.e. *NYC* and *Cainiao*. The *NYC* dataset [53] was collected from two types of taxis (yellow and green) in New York City, USA, and has been widely used as a benchmark in ridesharing studies [25], [27]. We select the day (April 09, 2016) with the largest amount of trips and extract data from it. It provides us the origins and destinations of $424,635$ trips, but does not provide the expected delivery time of them and the information of workers. Thus, we randomly generate the workers' initial locations and set the speed and capacity of each worker as $11.11m/s$ (about $40km/h$) and 4 respectively. We set the expected delivery time ratio (edt ratio for short) μ as a parameter in the experiment. The expected delivery time is set as $(1 + \mu)$ times the travel time needed to complete the task. The *Cainiao* dataset was collected by the largest delivery platform called Cainiao [8] in China, and was published by a publicly available last-mile delivery dataset with millions of packages from industry



(a) Total Tardiness (b) Total Travel Time (c) Total Latency (d) Makespan (e) Running Time

(f) Total Tardiness (g) Total Travel Time (h) Total Latency (i) Makespan (j) Running Time

Fig. 7. Results of NYC Dataset when Varying $|T|$ and $|W|$.

called LaDe [54]. It provides the daily information of tasks and workers in the Cainiao platform during 2021 with all information we needed. We select 147,350 actual delivery task data in Shanghai, including the initial location distribution of the workers. For both datasets, we vary the number of workers $|W|$ and the batch size Δt to simulate different situations in reality. For *NYC* dataset, we also vary the number of tasks $|T|$, the edt ratio μ to evaluate the performances. The parameter settings of *NYC* and *Cainiao* datasets are listed in table III and table IV respectively. The default settings are marked in bold.

B. Approaches and Measurements

We compare with the following algorithms in batch mode:

- **GreedyDP** [25] (**GDP** for short) is an insertion-based algorithm. Specifically, for each task, it will greedily select the worker with minimal increased travel time and inserts the origin and destination of the task into the optimal position of the worker's existing route.
- **FESI** [26] uses a travel budget to bound the longest travel time of workers. Initially, the travel budget is set to a small constant (e.g., 1), then in each iteration, it is doubled and more tasks can be assigned to workers. The algorithm terminates when all tasks have been assigned. Under each travel budget, it plans routes for the assigned tasks and append them to the existing routes of workers. It leverages the spatial index HST to do route planning and prove that the solution has an approximation ratio $O(\log n)$ in terms of both makespan and total latency. The implementation is open-source, but it only considers task assignment and route planning in one batch. We extend it to support multiple batches.
- **TrendSharing** (**TS** for short) is our method. Firstly, we sort the tasks with their *k-regret* values, then iterate each unassigned task, find the worker who can deliver it with minimal tardiness. Then, we search the largest trend that contains the task and the worker can accommodate from the flow tree and assign tasks in the trend to the chosen worker. After the assignment step, we plan a route for the tasks in the trend and append it to the worker's existing route.

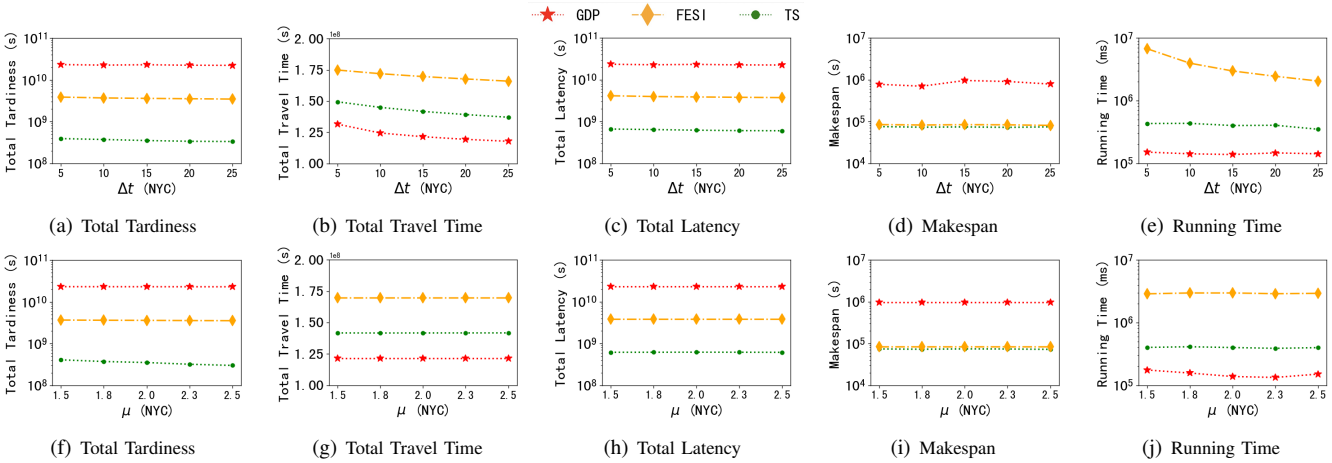
The following five metrics are used to evaluate the performances of the algorithms.

- **Total Tardiness**. The main objective of our work.
- **Total Travel Time**. The most common objective in existing works.
- **Total Latency**. The sum of the difference between the release time and completion time of all tasks. It includes the queueing time of tasks in batches and the process time.
- **Makespan**. The maximal travel time of all workers.
- **Running Time**. The time needed to complete task assignments and route plannings for all batches.

We implement all the compared algorithms in Java 11. All experiments are conducted on a server with two Intel(R) Xeon(R) 4210R 2.40 GHz CPUs and 128 GB memory.

C. Results of NYC Dataset

Effect of $|T|$. The first row in Figure 7 shows the results when varying the number of tasks $|T|$. The results of the algorithms increase as the number of tasks increases. In terms of total tardiness, TS significantly outperforms the state-of-the-art algorithms FESI and GDP by a factor about 7.2-10.2x and 43.9-65.6x, respectively. In terms of total travel time, GDP performs the best, TS comes second and the ratio between them is no more than 1.2. Although this is not the main objective of TS, it is still up to 1.2x smaller than FESI. The result of total latency for each algorithm is similar to that of total tardiness. The reason is, for each task, latency refers to the time interval between its completion time and its release time, while tardiness refers to the time interval between its completion time and its expected delivery time. Normally, the expected delivery time is larger than the release time. As a result, in the case when most tasks are tardy, these two metrics become similar. And when the expected delivery time is equal to the release time, total tardiness is equivalent to total latency. In terms of makespan, which is one of the objectives of FESI, TS still performs a bit better than FESI. GDP performs the worst. The reason is it will continuously insert tasks to the existing routes of workers. Workers in the area with a large number of tasks will be selected with high probability, while others in the area with few tasks may be in an idle state for a long time until a task traveling from the sparse area to the dense area appears. Consequently, the number of workers actually available is much less than the total number of workers, which will result in a worse makespan. In terms of



(a) Total Tardiness (b) Total Travel Time (c) Total Latency (d) Makespan (e) Running Time

(f) Total Tardiness (g) Total Travel Time (h) Total Latency (i) Makespan (j) Running Time

Fig. 8. Results of NYC Dataset when Varying Δt and μ.

running time, GDP performs the best and TS comes second. Although FESI needs a relatively long time to terminate, its response time will not exceed the batch period.

Effect of $|W|$. The second row in Figure 7 shows the results when varying the number of workers $|W|$. With the increasing number of workers, the results of all algorithms decrease except the running time. In terms of total tardiness, TS has a significant decrease when the number of workers increases from 1000 to 3000. The reason is when $|W| = 1000$, the number of workers is too small to meet actual demand. When $|W|$ increases to 3000, the relation between supply and demand is much more balanced and tardiness reduced by increasing workers is getting less and less. In terms of total travel time, we can see that it is not greatly affected by the number of workers. The reason is, the number of tasks is large and the empty run time of workers is negligible compared with the time needed to deliver tasks. In terms of total latency, we can observe that the gap between total tardiness and total latency of TS is clear, which means the proportion of tardiness in latency is relatively small. In terms of makespan, the results of TS and FESI are the best and they are close to each other. In terms of running time, the results of all three algorithms increase as the number of workers increases because the time complexities of all of them are affected by $|W|$.

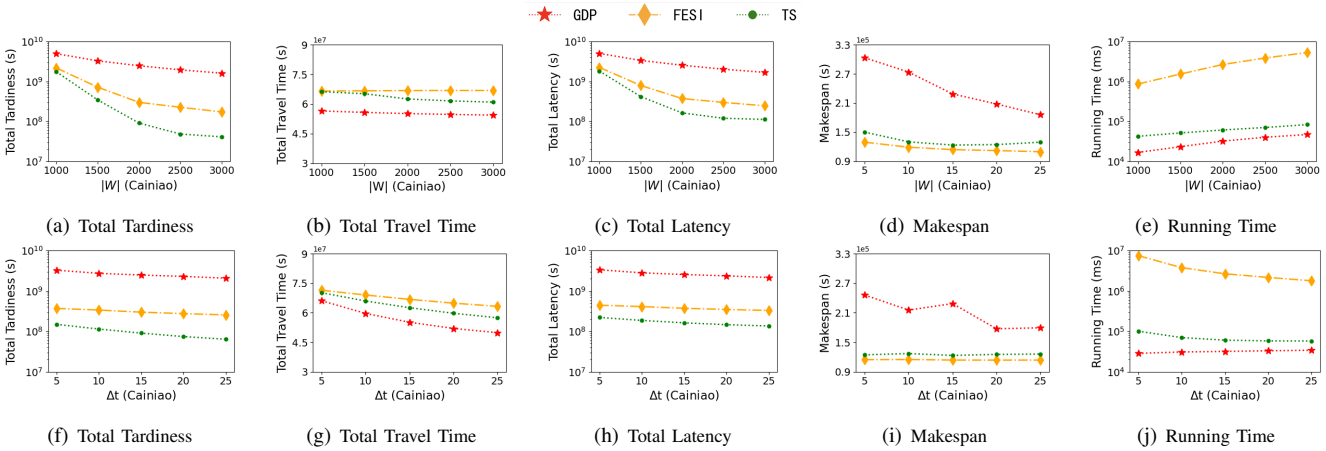
Effect of Δt. The first row in Figure 8 shows the results when varying the batch size Δt. We can observe that it has little impact of total tardiness and total latency. In terms of total travel time, the results of all three algorithms decrease. This is in line with expectations because despite the longer queuing time increasing the latency and indirectly affecting the tardiness, more tasks in a batch will lead to a better assignment. In terms of makespan, the performance of TS and FESI is stable, but the result of GDP has a little fluctuation. The running time of FESI is significantly reduced as the batch size increases because the number of tasks in each batch increases leading to a lower total cost of building HST.

Effect of μ. The second row in Figure 8 shows the results when varying the edt ratio μ. This parameter has little effect on the results of all three algorithms in terms of total travel time, total latency and makespan. For FESI and GDP, they do not consider the expected delivery time at all, so their performances will not be affected as well. For TS, this parameter will not influence the likelihood of tardiness for tasks. Thus, its result will not be affected. The total tardinesses of all three algorithms decrease with the increase of μ, and the reduction of TS is more obvious than the other two algorithms.

D. Results of Cainiao Dataset

Effect of $|W|$. The first row in Figure 9 shows the results when varying the number of workers $|W|$. In terms of total tardiness, the results of all algorithms decrease as the number of workers increases. TS is about 1.25-4.68x and 2.84-39.31x smaller than FESI and GDP, respectively. In terms of total travel time, GDP performs best, and TS is the runner up, the ratio between GDP and TS is no more than 1.17. Although this is not the main objective of TS, it is still up to 1.1x smaller than FESI. In terms of total latency, since the number of tardy tasks is less, the trend is no longer completely consistent with the result of total latency. The gap between TS and other algorithms is significantly reduced, TS is about 1.24x-2.5x and 2.76-16.95x smaller than FESI and GDP, which is a minor improvement compared with the result of total tardiness. This is a validation that algorithms that perform well on the objective of minimizing total latency may not perform well on the objective of minimizing total latency, although they look similar. In terms of makespan, the results of TS and FESI are close to each other, FESI shows a slight advantage here due to its use of travel budget. The result of GDP reduces significantly with the increasing number of workers. The reason is more workers appear in the area with a large number of tasks, so they can alleviate the burden of the overwhelmed workers. In terms of running time, GDP is the best since it is a greedy algorithm thus relatively faster. TS comes second and FESI is in the 3rd place, but are still adequate to respond promptly to users in real-world scenarios.

Effect of Δt. The second row in Figure 9 shows the result when varying batch size Δt. In terms of total tardiness, total latency and running time, the performance of the three algorithms appears to remain relatively stable. In terms of total travel time, the results of all three algorithms show a considerable downward trend as the batch size increases. TS gradually increased the gap with FESI in larger batches, demonstrating that our algorithm can effectively facilitate task sharing while ensuring minimal tardiness. GDP is committed to minimizing



(a) Total Tardiness (b) Total Travel Time (c) Total Latency (d) Makespan (e) Running Time

(f) Total Tardiness (g) Total Travel Time (h) Total Latency (i) Makespan (j) Running Time

Fig. 9. Results of Cainiao Dataset when Varying $|W|$ and Δt.

total travel time, which may lead to an uneven distribution among workers, and then fluctuations in the makespan.

E. Summary

TS v.s. GDP. TS outperforms the insertion-based algorithm GDP about 59.5x on *NYC* dataset and 39.3x on *Cainiao* dataset averagely in terms of total tardiness. And for the other metrics except for the total travel time and running time, TS significantly outperforms GDP. For the three algorithms that can respond in real-time, GDP possesses the absolute advantage over other algorithms for the total travel time metric because of the power of insertion operation. However, insertion may lead to the starvation of workers away from hotspots because the detour of a nearby worker may be much smaller than that of a far-away worker. Thus, workers in the suburb may stay idle for a long time. What's more, it will postpone the completion time of the tasks assigned before. These are the two reasons why its performance is poor for the other metrics. At the same time, it may cause the problem of uneven distribution among workers.

TS v.s. FESI. TS outperforms the state-of-the-art algorithm FESI about 9x on *NYC* dataset and 4.7x on *Cainiao* dataset averagely in terms of total tardiness. In terms of total latency, which is one of the two objectives of FESI, TS still performs better than FESI. In terms of makespan, which is the other objective of FESI, the performance of TS is close to that of FESI. Although FESI can bound the longest total travel time of workers by the travel budget, when the budget is small, the gap between the shortest route and longest route can be bound by a small interval. However, when the budget becomes large (e.g., $2^{16} = 65536$), the interval is large and cannot guarantee that tasks are assigned evenly to workers.

VIII. Conclusion

In this paper, we consider tardiness as the optimization objective and propose the minimum tardiness task assignment and route planning problem. To solve this problem, we propose a unified framework termed TrendSharing, which can mine the spatial characteristics of tasks and capture the temporal characteristics to well guide the task assignment and route planning procedure. Specifically, we propose a novel structure called flow tree to group tasks together according to their spatial features. To find a set of tasks with high sharability, we propose a concept of trend and use a decision factor ϵ-*score* to discover trend from the flow tree. In the task assignment step, we devise an indicator *k-regret* to quantify the likelihood of tardiness for each task and determine the order of tasks being selected. Then, we take the advantage of the trend and design an effective greedy algorithm to conduct task assignment. In the route planning step, we adopt a simple yet effective strategy to continuously append newly planned routes to the workers' existing routes. Moreover, we propose an algorithm to plan a route for the tasks in a trend with an approximation ratio 2.5 to the optimum. The experiment results validate that our methods can achieve a huge enhancement on the objective of minimizing total tardiness, and also perform well on other common optimization objectives.

IX. Acknowledgment

Peng Cheng's work is supported by the National Natural Science Foundation of China under Grant No. 62102149. Libin Zheng is supported by the National Natural Science Foundation of China No. 62102463 and the Natural Science Foundation of Guangdong Province of China No. 2022A1515011135. Lei Chen's work is partially supported by National Key Research and Development Program of China Grant No. 2023YFF0725100, National Science Foundation of China (NSFC) under Grant No. U22B2060, the Hong Kong RGC GRF Project 16213620, RIF Project R6020-19, AOE Project AoE/E-603/18, Theme-based project TRS T41-603/20R, CRF Project C2004-21G, Hong Kong ITC ITF grants MHX/078/21 and PRP/004/22FX, Microsoft Research Asia Collaborative Research Grant and HKUST-Webank joint research lab grants. Chen Jason Zhang is partially supported by PolyU (UGC) P0045695, ITF-ITSP P0043294, ITS/028/22FP, ITF PRP/009/22FX, PolyU-MinshangCT Generative AI Laboratory Fund No. P0046453, Research Matching Grant Scheme Funds No. P0048191 and No. P0048183, PolyU Start-up Fund No. P0046703. Xuemin Lin is supported by NSFC U2241211, NSFC U20B2046, and 23H020101910. Wenjie Zhang is supported by ARC DP230101445 and FT210100303. Corresponding author: Peng Cheng.

REFERENCES

[1] "Uber." https://www.uber.com/.
[2] "Didi." https://www.didiglobal.com/.
[3] "Lyft." https://www.lyft.com/.
[4] "Grubhub." https://www.grubhub.com/.
[5] "Meituan." https://www.meituan.com/.
[6] "Eleme." https://www.ele.me/.
[7] "UPS." https://www.ups.com/.
[8] "Cainiao." https://www.cainiao.com/.
[9] L. Kazemi and C. Shahabi, "Geocrowd: enabling query answering with spatial crowdsourcing," in *Proceedings of the 20th international conference on advances in geographic information systems*, pp. 189–198, 2012.
[10] L. Kazemi, C. Shahabi, and L. Chen, "Geotrucrowd: trustworthy query answering with spatial crowdsourcing," in *Proceedings of the 21st acm sigspatial international conference on advances in geographic information systems*, pp. 314–323, 2013.
[11] H. To, C. Shahabi, and L. Kazemi, "A server-assigned spatial crowdsourcing framework," *ACM Transactions on Spatial Algorithms and Systems (TSAS)*, vol. 1, no. 1, pp. 1–28, 2015.
[12] D. Deng, C. Shahabi, and L. Zhu, "Task matching and scheduling for multiple workers in spatial crowdsourcing," in *Proceedings of the 23rd SIGSPATIAL International Conference on Advances in Geographic Information Systems*, pp. 1–10, 2015.
[13] M. Asghari, D. Deng, C. Shahabi, U. Demiryurek, and Y. Li, "Price-aware real-time ride-sharing at scale: an auction-based approach," in *Proceedings of the 24th ACM SIGSPATIAL international conference on advances in geographic information systems*, pp. 1–10, 2016.
[14] M. Asghari and C. Shahabi, "An on-line truthful and individually rational pricing mechanism for ride-sharing," in *Proceedings of the 25th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, pp. 1–10, 2017.
[15] L. Zheng, L. Chen, and J. Ye, "Order dispatch in price-aware ridesharing," *Proceedings of the VLDB Endowment*, vol. 11, no. 8, pp. 853–865, 2018.
[16] Y. Zeng, Y. Tong, Y. Song, and L. Chen, "The simpler the better: an indexing approach for shared-route planning queries," *Proceedings of the VLDB Endowment*, vol. 13, no. 13, pp. 3517–3530, 2020.
[17] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to algorithms*. MIT press, 2009.
[18] H. To, L. Fan, L. Tran, and C. Shahabi, "Real-time task assignment in hyperlocal spatial crowdsourcing under budget constraints," in *2016 IEEE International Conference on Pervasive Computing and Communications (PerCom)*, pp. 1–8, IEEE, 2016.
[19] P. Cheng, X. Lian, L. Chen, and C. Shahabi, "Prediction-based task assignment in spatial crowdsourcing," in *Data Engineering (ICDE), 2017 IEEE 33rd International Conference on*, pp. 997–1008, IEEE, 2017.
[20] J.-J. Jaw, A. R. Odoni, H. N. Psaraftis, and N. H. Wilson, "A heuristic algorithm for the multi-vehicle advance request dial-a-ride problem with time windows," *Transportation Research Part B: Methodological*, vol. 20, no. 3, pp. 243–257, 1986.
[21] S. Ma, Y. Zheng, and O. Wolfson, "T-share: A large-scale dynamic taxi ridesharing service," in *2013 IEEE 29th International Conference on Data Engineering (ICDE)*, pp. 410–421, IEEE, 2013.
[22] S. Ma, Y. Zheng, and O. Wolfson, "Real-time city-scale taxi ridesharing," *IEEE Transactions on Knowledge and Data Engineering*, vol. 27, no. 7, pp. 1782–1795, 2014.
[23] Y. Huang, F. Bastani, R. Jin, and X. S. Wang, "Large scale real-time ridesharing with service guarantee on road networks," *Proceedings of the VLDB Endowment*, vol. 7, no. 14, 2014.
[24] 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 *2017 IEEE 33rd International Conference on Data Engineering (ICDE)*, pp. 1117–1128, IEEE, 2017.
[25] Y. Tong, Y. Zeng, Z. Zhou, L. Chen, J. Ye, and K. Xu, "A unified approach to route planning for shared mobility," *Proceedings of the VLDB Endowment*, vol. 11, no. 11, p. 1633, 2018.
[26] Y. Zeng, Y. Tong, and L. Chen, "Last-mile delivery made practical: An efficient route planning framework with theoretical guarantees," *Proceedings of the VLDB Endowment*, vol. 13, no. 3, pp. 320–333, 2019.
[27] J. Wang, P. Cheng, L. Zheng, C. Feng, L. Chen, X. Lin, and Z. Wang, "Demand-aware route planning for shared mobility services," *Proceedings of the VLDB Endowment*, vol. 13, no. 7, pp. 979–991, 2020.
[28] G. B. Dantzig and J. H. Ramser, "The truck dispatching problem," *Management science*, vol. 6, no. 1, pp. 80–91, 1959.
[29] G. Laporte, "The vehicle routing problem: An overview of exact and approximate algorithms," *European journal of operational research*, vol. 59, no. 3, pp. 345–358, 1992.
[30] B. L. Golden, S. Raghavan, E. A. Wasil, *et al.*, *The vehicle routing problem: latest advances and new challenges*, vol. 43. Springer, 2008.
[31] G. Berbeglia, J.-F. Cordeau, I. Gribkovskaia, and G. Laporte, "Static pickup and delivery problems: a classification scheme and survey," *Top*, vol. 15, no. 1, pp. 1–31, 2007.
[32] G. Berbeglia, J.-F. Cordeau, and G. Laporte, "Dynamic pickup and delivery problems," *European journal of operational research*, vol. 202, no. 1, pp. 8–15, 2010.
[33] J.-F. Cordeau and G. Laporte, "The dial-a-ride problem: models and algorithms," *Annals of operations research*, vol. 153, no. 1, pp. 29–46, 2007.
[34] A. Colorni and G. Righini, "Modeling and optimizing dynamic dial-a-ride problems," *International transactions in operational research*, vol. 8, no. 2, pp. 155–166, 2001.
[35] J.-F. Cordeau, "A branch-and-cut algorithm for the dial-a-ride problem," *Operations Research*, vol. 54, no. 3, pp. 573–586, 2006.
[36] P. Shaw, "A new local search algorithm providing high quality solutions to vehicle routing problems," *APES Group, Dept of Computer Science, University of Strathclyde, Glasgow, Scotland, UK*, vol. 46, 1997.
[37] S. Ropke and D. Pisinger, "An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows," *Transportation science*, vol. 40, no. 4, pp. 455–472, 2006.
[38] D. Pisinger and S. Ropke, "Large neighborhood search," in *Handbook of metaheuristics*, pp. 99–127, Springer, 2019.
[39] J.-F. Cordeau and G. Laporte, "A tabu search heuristic for the static multi-vehicle dial-a-ride problem," *Transportation Research Part B: Methodological*, vol. 37, no. 6, pp. 579–594, 2003.
[40] H. Yuan and G. Li, "A survey of traffic prediction: from spatio-temporal data to intelligent transportation," *Data Science and Engineering*, vol. 6, no. 1, pp. 63–85, 2021.
[41] H. Emmons, "One-machine sequencing to minimize certain functions of job tardiness," *Operations Research*, vol. 17, no. 4, pp. 701–715, 1969.
[42] E. L. Lawler, "A "pseudopolynomial" algorithm for sequencing jobs to minimize total tardiness," in *Annals of discrete Mathematics*, vol. 1, pp. 331–342, Elsevier, 1977.
[43] C. Koulamas, "The total tardiness problem: review and extensions," *Operations research*, vol. 42, no. 6, pp. 1025–1041, 1994.
[44] C. Koulamas, "The single-machine total tardiness scheduling problem: Review and extensions," *European journal of operational research*, vol. 202, no. 1, pp. 1–7, 2010.
[45] Y. Tong, Z. Zhou, Y. Zeng, L. Chen, and C. Shahabi, "Spatial crowdsourcing: a survey," *The VLDB Journal*, vol. 29, no. 1, pp. 217–250, 2020.
[46] S. O. Krumke, "Online optimization: Competitive analysis and beyond," 2002.
[47] Y. Bartal, "Probabilistic approximation of metric spaces and its algorithmic applications," in *Proceedings of 37th Conference on Foundations of Computer Science*, pp. 184–193, IEEE, 1996.
[48] J. Fakcharoenphol, S. Rao, and K. Talwar, "A tight bound on approximating arbitrary metrics by tree metrics," *Journal of Computer and System Sciences*, vol. 69, no. 3, pp. 485–497, 2004.
[49] A. Backurs, P. Indyk, K. Onak, B. Schieber, A. Vakilian, and T. Wagner, "Scalable fair clustering," in *International Conference on Machine Learning*, pp. 405–413, PMLR, 2019.
[50] B. Behsaz, Z. Friggstad, M. R. Salavatipour, and R. Sivakumar, "Approximation algorithms for min-sum k-clustering and balanced k-median," *Algorithmica*, vol. 81, no. 3, pp. 1006–1030, 2019.
[51] M. R. Garey and D. S. Johnson, *Computers and intractability*, vol. 174. freeman San Francisco, 1979.
[52] N. Christofides, "Worst-case analysis of a new heuristic for the travelling salesman problem," tech. rep., Carnegie-Mellon Univ Pittsburgh Pa Management Sciences Research Group, 1976.
[53] "NYC dataset." https://www1.nyc.gov/site/tlc/about/tlc-trip-record-data.page.
[54] "Cainiao dataset." https://huggingface.co/datasets/Cainiao-AI/LaDe-D.