Towards Minimum Fleet for Ridesharing-Aware Mobility-on-Demand Systems

Chonghuan Wang[†], Yiwen Song[†], Yifei Wei[§], Guiyun Fan[†], Haiming Jin[*‡], Fan Zhang[¶]
[†]Department of Electronic Engineering, Shanghai Jiao Tong University, Shanghai, China
[‡]John Hopcroft Center for Computer Science, Shanghai Jiao Tong University, Shanghai, China
[§]Department of Computer Science and Engineering, Shanghai Jiao Tong University, Shanghai, China
[¶]Shenzhen Institutes of Advanced Technology, Chinese Academy of Sciences, Shenzhen, China
Email: {wangchonghuan, gavinsyw, weiyifei, fgy726, jinhaiming}@sjtu.edu, zhangfan@siat.ac.cn

***Abstract*—The rapid development of information and communication technologies has given rise to mobility-on-demand (MoD) systems (e.g., Uber, Didi) that have fundamentally revolutionized urban transportation. One common feature of today's MoD systems is the integration of *ridesharing* due to its cost-efficient and environment-friendly natures. However, a fundamental unsolved problem for such systems is how to serve people's heterogeneous transportation demands with as few vehicles as possible. Naturally, solving such minimum fleet problem is essential to reduce the vehicles on the road to improve transportation efficiency. Therefore, we investigate *the fleet minimization problem in ridesharing-aware MoD systems*. We use graph-theoretic methods to construct a novel *order graph* capturing the complicated inter-order shareability, each order's spatial-temporal features, and various other real-world factors. We then formulate the problem as a *tree cover* problem over the order graph, which differs from the traditional coverage problems. Theoretically, we prove the problem is *NP-hard*, and propose a *polynomial-time* algorithm with a guaranteed approximation ratio. Besides, we address the *online fleet minimization* problem, where orders arrive in an online manner. Finally, extensive experiments on a city-scale dataset from Shenzhen, containing 21 million orders from June 1st to 30th, 2017, validate the effectiveness of our algorithms.**

I. Introduction

With the rapid development of mobile Internet, cloud computing, and sensing technologies, *mobility-on-demand (MoD)* systems (e.g., Uber, Lift, Didi) have fundamentally revolutionized the way urban residents travel and commute [1–4]. In a typical MoD system, passengers send via a smartphone app their ride orders to a platform, which usually resides in the cloud, keeps tracking the GPS coordinates of the vehicles managed by it, and matches the vast number of passenger orders with the available vehicles in real time. Foreseeably, the promising rise of the increasingly capable autonomous and connected vehicles will further facilitate such way of on-demand urban mobility in the near future.

Nowadays, existing MoD systems invariably choose to integrate *ridesharing* into their services by offering passengers the flexible choice of sharing a ride with others, which oftentimes costs less than taking a ride by themselves. Besides cost efficiency, through enabling one vehicle to serve multiple orders simultaneously, ridesharing could also potentially dramatically decrease the overall number of vehicles on the road, which consequently helps conserve non-renewable energy resources, reduce air pollution, and alleviate traffic congestion. As a result of the above benefits, *ridesharing awareness* has thus become a common feature of today's MoD systems.

[*]Corresponding author.

Though promising, to serve people's massive demands for personal mobility, existing MoD systems still have to run thousands of vehicles even in a single city, which constitute a significant portion of the overall number of vehicles a city has. It thus becomes imperative to seek ways so as to best size and operate the fleet of on-demand vehicles for transportation efficiency and environment friendliness. Therefore, in this paper, we investigate the problem of *minimizing the vehicle fleet needed to serve people's mobility demands in ridesharing-aware MoD systems*. Although the minimum fleet problem has already been solved for ridesharing-free MoD systems by a pioneering work in Nature [5], the ridesharing awareness, together with the diverse and complicated scenarios of real-world MoD systems, poses a wide scope of unique challenges that cannot be fully addressed by the methods proposed in [5], as well as other existing works.

The first challenge comes from the *heterogeneity of mobility demands* in ridesharing-aware MoD systems, such as whether the passengers are willing to share a vehicle, how many passengers one order has, when and where the passengers want to be picked up, and many others. Such diverse heterogeneity inevitably makes it highly non-trivial to construct a realistic model for ridesharing-aware MoD systems. Next, the *more complicated shareability relationships* among orders caused by the integration of ridesharing further aggravate the challenges of making appropriate order-vehicle matching decisions. In a ridesharing-free MoD system, a vehicle can only be shared by passenger orders across different time and locations. However, ridesharing-aware MoD systems additionally allow multiple orders to share one vehicle at the same time, where an order may even be shareable with different sets of orders as time evolves. Furthermore, minimizing the city-wide on-demand vehicle fleet with *humongous numbers of passenger orders* is naturally a large-scale optimization problem, which inevitably calls for efficient solution algorithms with low computational complexity so as to be implementable in practice.

To address these challenges, we build a novel *order graph* for ridesharing-aware MoD systems based on graph-theoretic methods, which captures collectively the spatial-temporal features of each order, the complex inter-order shareability relationships, as well as a wide spectrum of other real-world factors. Under such graph representation, the set of orders that each vehicle serves naturally form a tree structure. We thus formulate the minimum fleet problem as finding the minimum number of trees that cover the entire order graph. Such *tree cover* problem is similar to yet fundamentally different from traditional coverage problems (e.g., submodular cover). Theoretically, we prove the NP-hardness of the formulated tree cover problem by a reduction from the traditional bin packing problem. We then design a polynomial-time algorithm to solve the problem with a guaranteed approximation ratio. Apart from the offline setting, we also investigate the *online fleet minimization* problem, where passenger orders arrive sequentially in an online manner, and propose an efficient algorithm for making online order dispatch decisions.

In summary, this paper makes the following contributions.

- To the best of our knowledge, this paper is the first work that addresses the city-scale *minimum fleet* problem for *ridesharing-aware* MoD systems.
- We build a novel *order graph* for ridesharing-aware MoD systems, which effectively captures the diverse heterogeneity and complex sharability among passenger orders. Besides, we formulate the minimum fleet problem as finding the minimum *tree cover* of the order graph, which fundamentally differs from the traditional coverage problems.
- Theoretically, we prove that the formulated tree cover problem is *NP-hard*, and propose a polynomial-time algorithm to solve it with a $1 + \ln \lambda$ *approximation ratio*, where λ is the largest number of orders that one vehicle serves.
- Extensive experiments are conducted on a city-scale real-world dataset of Shenzhen, China, containing 21 million passenger orders from June 1st to June 30th, 2017, which validate the effectiveness of our algorithms.

II. Preliminaries

In this section, we give an overview of MoD systems, define the order graph, and describe the problem we aim to solve.

A. System Overview

We consider a ridesharing-aware MoD system operated by a cloud-based platform, which tracks the locations of the vehicles in the system by collecting their GPS coordinates every few seconds, and receives the ride orders that passengers send via their smartphones. In the back-end, the platform periodically executes the order dispatch algorithm [6–9] to match passenger orders with available vehicles, which may dispatch multiple orders willing to share their rides to the same vehicle at the same time. After the vehicle-order matching, the platform will then return the matching results to the smartphones of the passengers and drivers in the system.

We discretize the entire geographical area into P equal-size non-overlapping cells, denoted as $\mathcal{P} = \{1, 2, \cdots, P\}$. Moreover, we consider cell[1] as the smallest unit for location, and do not distinguish the locations within a cell. Similarly, we discretize the planning horizon into T equal-length (e.g., 10 seconds) time slots, denoted as $\mathcal{T} = \{1, 2, \cdots, T\}$. Based on $\mathcal{P}$ and $\mathcal{T}$, we define a *trajectory* in Definition 1.

Definition 1 (Trajectory). *A length-l trajectory, denoted as $\varphi = \{(t_1, p_1), (t_2, p_2), \cdots, (t_l, p_l)\}$, is a series of l spatial-temporal tuple, where each tuple (t_k, p_k) of φ satisfies that $(t_k, p_k) \in \mathcal{T} \times \mathcal{P}$, t_{k+1} and t_k satisfies that $t_{k+1} = t_k + 1$, and p_k is the cell that the trajectory traverses at time slot t_k.*

Note that the length of a trajectory refers to the number of consecutive time slots it lasts rather than its physical length.

B. Order Graph

In our MoD system, passengers can choose whether to share rides with others. We thus define an *order* as in Definition 2.

Definition 2 (Order). *We define an order in our MoD system as a 3-tuple $o_i = (\varphi_i, n_i, \xi_i)$.*

- *$\varphi_i = \{(t_{i,1}, p_{i,1}), (t_{i,2}, p_{i,2}), \cdots, (t_{i,l_i}, p_{i,l_i})\}$ denotes the trajectory of this order that starts from the time slot $t_{i,1}$when the passegers of this order are picked up, and ends at the time slot t_{i,l_i} when they arrive at their destination.*
- *n_i represents the number of passengers of o_i. We use C to denote the maximum number of passengers that a vehicle could carry, and thus, $n_i \leq C$.*
- *$\xi_i \in \{0, 1\}$ indicates whether o_i is a ridesharing order. That is, $\xi_i = 1$ indicates that order o_i is willing to share a vehicle with others, and $\xi_i = 0$ otherwise.*

With the advancement of machine learning techniques, given the origin, destination, and starting time, the trajectory of an order could be well estimated using various methods, such as diffusion convolutional recurrent neural network [10], graph multi-attention network [11], and many others [12,13]. Thus, in our model, each order o_i uniquely corresponds to a trajectory φ_i. We use $\varphi_i(t_1, t_2)$ to denote the part of trajectory φ_i from time slot t_1 to t_2. Next, in the following Definition 3, we define the concept of *shareable orders*.

Definition 3 (Shareable Orders). *Two orders o_i and o_j are shareable (i.e., can share a vehicle simultaneously), if they are both ridesharing orders (i.e., $\xi_i = \xi_j = 1$), and either of the two shareability conditions (s-Conditions) is satisfied.*

- ***s-Condition 1:*** *The trajectory of either o_i or o_j is a subset of the other's (i.e., $\varphi_i \subset \varphi_j$ or $\varphi_j \subset \varphi_i$), and $n_i + n_j \leq C$.*
- ***s-Condition 2:*** *o_i and o_j satisfy $n_i + n_j \leq C$, and the following Equation (1),*

$$\begin{cases} \varphi_i(t_{i,1}, t_{j,l_j}) = \varphi_j(t_{i,1}, t_{j,l_j}), & \text{if } t_{i,1} > t_{j,1}, \\ \varphi_i(t_{j,1}, t_{i,l_i}) = \varphi_j(t_{j,1}, t_{i,l_i}), & \text{otherwise}, \end{cases} \quad (1)$$

which ensures that if order o_j starts before order o_i (i.e., $t_{i,1} > t_{j,1}$), the trajectories of them overlap between time slots $t_{i,1}$ and t_{j,l_j}, and otherwise, their trajectories overlap between time slots $t_{j,1}$ and t_{i,l_i}.

Naturally, three or more orders are shareable, if each pair of them are shareable, and they have a total number of passengers upper bounded by C. Figure 1 shows an example of two pairs

[1]The size of a cell can be chosen flexibly in our model, and as will be discussed after Definition 5, it does not affect the scalability of our model.

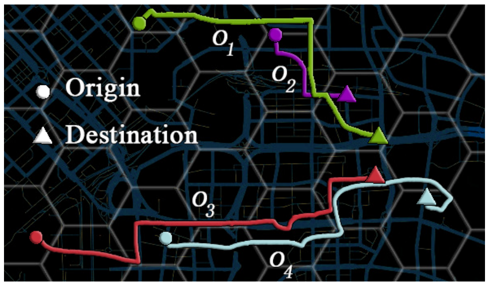

Fig. 1: Example of shareable orders, where different orders traverse the same cell in the same time slot.

of shareable orders, where orders o_1 and o_2 satisfy s-Condition 1, because o_2's trajectory is a subset of o_1's, and orders o_3 and o_4 satisfy s-Condition 2, as their trajectories partly overlap.

Besides shareability, *succession* is another relationship that widely exists among orders, which we define in Definition 4.

Definition 4 (Order Succession)**.** *Order o_j is successional to o_i (i.e., can be served in succession to o_i), if a vehicle can reach o_j's origin $p_{j,1}$ by its starting time $t_{j,1}$ after serving o_i.*

Next, we formally define the *order graph* in Definition 5.

Definition 5 (Order Graph)**.** *The order graph of any order set $\mathcal{O}$ is a directed graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where each vertex $v_{i,m} \in \mathcal{V}$ corresponds to the spatial-temporal tuple $(t_{i,m}, p_{i,m})$ of order $o_i \in \mathcal{O}$, and there is a directed edge $(v_{i,m}, v_{j,n})$ from $v_{i,m}$ to $v_{j,n}$, if any of the three edge conditions (e-Conditions) holds.*

- ***e-Condition 1** (Intra-Order): $v_{i,m}$ and $v_{j,n}$ are from the same order and adjacent in time (i.e., $i = j$ and $n = m+1$).*
- ***e-Condition 2** (Shareable Orders): o_i and o_j are shareable, o_j starts at time slot $t_{i,m}$ (i.e., $t_{i,m} = t_{j,1}$), and $v_{j,n}$ corresponds to o_j's origin (i.e., $n = 1$).*
- ***e-Condition 3** (Order Succession): o_j is successional to o_i, $v_{i,m}$ corresponds to o_i's destination (i.e., $m = l_i$), and $v_{j,n}$ corresponds to o_j's origin (i.e., $n = 1$).*

That is, three types of edges may exist in an order graph, including those between two consecutive spatial-temporal tuples of the same order, and those from one order to another shareable with or successional to it. Note that the cell size will not harm the scalability of the order graph. In fact, the smaller a cell, the stricter the constraint is for two orders to be shareable, and thus the less edges an order graph will have.

Table I gives an example of five orders in an MoD system, and Figure 2 shows the corresponding order graph with $C = 3$. By Definition 5, each vertex $v_{i,m}$ stands for the mth spatial-temporal tuple of order o_i. Furthemore, there exist edges between vertices of the same order as they satisfy e-Condition 1, edges $(v_{1,2}, v_{2,1})$, $(v_{1,3}, v_{3,1})$, and $(v_{2,2}, v_{3,1})$ exist due to e-Condition 2, and e-Condition 3 leads to the existence of edges $(v_{1,7}, v_{5,1})$, $(v_{3,2}, v_{4,1})$, and $(v_{2,4}, v_{4,1})$. Although the trajectories of o_1 and o_4 overlap, no edge exists from $v_{1,6}$ to $v_{4,1}$, because o_4 is not a ridesharing order (i.e., $\xi_4 = 0$).

C. Problem Description

As aforementioned, this paper aims to *minimize the number of vehicles needed to serve the passenger orders in a ridesharing-aware MoD system.* We start addressing this problem by formally describing it as an optimization over the order graph. As building a complete order graph needs all the orders' information, the problem described here is the offline version. Its online version will be described in Section V.

Order	φ_i	n_i	ξ_i
o_1	$\{(t_1,l_1),(t_2,l_2),(t_3,l_3),(t_4,l_4),(t_5,l_5),(t_6,l_6),(t_7,l_7)\}$	2	1
o_2	$\{(t_2,l_2),(t_3,l_3),(t_4,l_4),(t_5,l_5)\}$	1	1
o_3	$\{(t_3,l_3),(t_4,l_4)\}$	1	1
o_4	$\{(t_6,l_{15}),(t_7,l_7),(t_8,l_8),(t_9,l_9),(t_{10},l_{10})\}$	1	0
o_5	$\{(t_9,l_{11}),(t_{10},l_{12}),(t_{11},l_{13}),(t_{12},l_{14})\}$	1	1

TABLE I: Example of five orders in an MoD system.

Next, we define the *dominating order* among those served by a vehicle at the same time as the one that finishes the last, which changes over time. For example, in Figure 2, if an empty vehicle picks up order o_3 at time slot t_3, o_3 is the vehicle's dominating order at t_3. At time slot t_6, when the same vehicle starts serving order o_4, its dominating order switches to o_4.

Clearly, all of the vertices of the same order forms a path in the order graph, and we thus refer to such path as an *order path*. Next, we define the *dispatching tree* in Definition 6.

Definition 6 (Dispatching Tree)**.** *A dispatching tree over an order graph is a tree that consists of multiple complete order paths that satisfy the following tree constraints (t-Constraints).*

- ***t-Constraint 1:** Any edge in a dispatching tree which satisfies e-Condition 2 or 3 starts from the dominating order.*
- ***t-Constraint 2:** At any time slot, the tree covers orders with at most C passengers in total.*

We use $\mathcal{D}$ to denote the set of all the dispatching trees over the order graph.

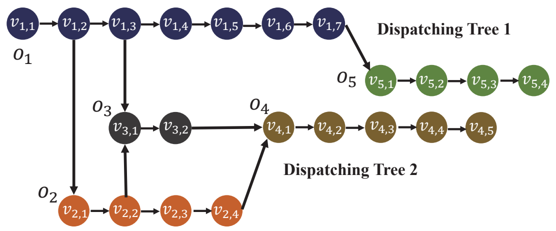

Fig. 2: Order graph that corresponds to Table I.

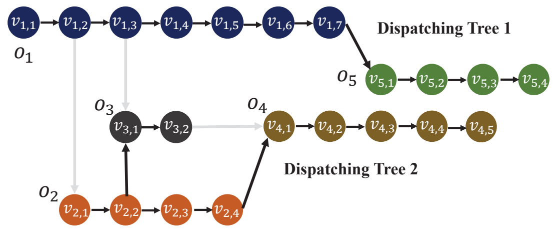

Fig. 3: Constructed dispatching trees over the order graph in Figure 2.

Essentially, t-Constraint 1 ensures that any order being served by a vehicle can maintain its trajectory until it finishes, even though the vehicle picks up a new ridesharing order. In Figure 3, we show two dispatching trees constructed over the order graph in Figure 2, where dispatching tree 1 consists of orders o_1 and o_5, and dispatching tree 2 contains orders o_2, o_3, and o_4. Clearly, orders o_1, o_3, and o_4 cannot form a feasible dispatching tree, as edge $(v_{3,2}, v_{4,1})$ violates t-Constraint 1.

By Definition 6, the orders in one dispatching tree can be served by one vehicle, and as an order can only be served once, each order path should be covered by exactly one dispatching tree. A dispatching scheme for all orders can thus be represented as a forest composed of multiple dispatching trees that cover the entire order graph.

Thus, the aforementioned minimum fleet problem is equivalent to the *dispatching tree cover (DTC)* problem given in Definition 7, which we actually solve in this paper.

Definition 7 (DTC Problem)**.** *Given an order graph, the DTC problem aims to find a forest $\mathcal{F}^* \subseteq \mathcal{D}$ with the minimum cardinality, referred to as the minimum dispatching tree (DT) cover, such that each order path is covered by one and only one dispatching tree in $\mathcal{F}^*$.*

Figure 3 shows the minimum DT cover of the order graph in Figure 2, $\mathcal{F}^* = \{\{o_1, o_5\}, \{o_2, o_3, o_4\}\}$, which means that the five orders in the example can be served by two vehicles.

III. Formulation and Hardness Analysis

A. Mathematical Formulation

We first formulate the DTC problem given in Definition 7 formally as the following integer-linear optimization program.

$$\textbf{DTC}: \min \sum_{i:\tau_i \in \mathcal{D}} x_i \tag{2}$$

$$\text{s.t.} \sum_{i:o_j \in \tau_i} x_i = 1, \forall o_j \in \mathcal{O}, \tag{2a}$$

$$x_i \in \{0,1\}, \quad \forall \tau_i \in \mathcal{D}. \tag{2b}$$

Although the above **DTC** program requires the knowledge of the dispatching tree set $\mathcal{D}$, whose size is exponential w.r.t. to the overall number of orders, such knowledge is merely needed for mathematically representing the DTC problem. In fact, the DTC problem itself, as well as the algorithms we propose in Section IV only need as input the order graph $\mathcal{G}$, instead of the dispatching tree set. Furthermore, in **DTC**, each dispatching tree $\tau_i \in \mathcal{D}$ corresponds to a binary variable $x_i \in \{0,1\}$, where $x_i = 1$ indicates that dispatching tree τ_i is selected in the DT cover, and otherwise τ_i is not selected. The objective function $\sum_{i:\tau_i \in \mathcal{D}} x_i$ given by Equation (2) is exactly the total number of selected dispatching trees. Constraint (2a) means each order is covered by exactly one dispatching tree.

The DTC problem, though shares some resemblance with the minimum submodular cover (MSC) problem [14], is fundamentally different from it. In DTC, an order can be covered once and only once, which indicates that the ground set (i.e., the set of all feasible dispatching trees $\mathcal{D}$) is not necessarily a feasible solution for DTC, and thus makes DTC violate submodularity. Specifically, DTC differs from the set cover (SC) problem, which is essentially an MSC problem, as each element in SC can be covered unlimited times. Moreover, the path cover problem [5] is clearly a special case of DTC. The above differences distinguish DTC from traditional coverage problems, and call for a new suite of solutions and analyses.

B. Hardness Analysis

In order to show the hardness of the DTC problem, we first define the *2-DTC problem*, which is a special case of the DTC problem with a stricter constraint that each vehicle can serve at most two orders at the same time. We refer to any dispatching tree that satisfies such constraint as a *width-2 dispatching tree*, and the set that consists of all the width-2 dispatching trees over the order graph as $\mathcal{D}_2$. Clearly, at any time slot, a width-2 dispatching tree has vertices from at most two orders. The mathematical formulation of 2-DTC is almost the same with that of DTC except that the set $\mathcal{D}$ is replaced by $\mathcal{D}_2$. Next, we state the decision version of 2-DTC as follows.

- **INSTANCE:** Order graph $\mathcal{G}$ and positive integer K.
- **QUESTION:** Can all the order paths in $\mathcal{G}$ be covered once and only once by at most K width-2 dispatching trees?

Next, we prove in Theorem 1 the NP-completeness of the above decision version of 2-DTC.

Theorem 1. *The decision version of the 2-DTC problem is NP-complete.*

Proof. We construct prove this theorem by a reduction from bin packing to 2-DTC. We let $\mathcal{I}$ be a finite set of items, $s(u_n) \in \mathbb{Z}^+$ be the size of each item $u_n \in \mathcal{I}$, and an integer B be the capacity of every bin. Given any positive integer K, the *bin packing problem* [15] asks whether $\mathcal{I}$ can be partitioned into disjoint sets $\mathcal{I}_1, \mathcal{I}_2, \cdots, \mathcal{I}_K$ such that the overall size of the items in each set does not exceed B. Clearly, we have $\sum_n s(u_n) \leq KB$. Note that if the capacity B goes to infinity, the bin packing problem can be directly solved. Thus, assuming B is upper bounded by a sufficiently large constant U (i.e., $B < U$) does not affect its hardness.

Next, we describe our construction of an order graph from an instance of bin packing. We use $s(u_n)$ and B vertices to represent each item and each bin, respectively. We denote the B vertices from bin i as $c_{i,B}, c_{i,B-1}, \cdots, c_{i,1}$, and introduce a directed edge from $c_{i,m}$ to $c_{i,m-1}$ for each $1 \leq i \leq K$ and $1 < m \leq B$ to form an order path. Similarly, we introduce directed edges to connect the vertices of each item to form an order path. Moreover, we connect $c_{i,j}$ to the first node of u_n's path, if $j \geq s(u_n)$ (i.e., the remaining capacity of bin i is enough to carry u_n). If a tree chooses the edge from $c_{i,m}$ to u_n, it means that bin i takes in u_n with remaining capacity m, and the tree cannot have other edges starting from $\{c_{i,m}, c_{i,m-1}, \cdots, c_{i,j-s(u_n)+1}\}$, as the capacity has been held by u_n. In this way, each feasible tree is thus width-2. Then, the problem becomes finding K width-2 trees to cover all the order paths. As there are no edges between the order paths of different bins, the K bins' order paths will surely be in different trees. Besides, as $B \geq s(u_n), \forall u_n \in \mathcal{I}$, the order paths of the bins are surely the dominating paths. Thus far, bin packing has been successfully reduced to 2-DTC.

The above reduction takes $\sum_n s(u_n) + KB$ steps to construct all the vertices, which is upper bounded by $2KB$. The complexity for constructing the edges cannot exceed $O(2KB + |\mathcal{I}|B)$, as there are no more than $O(2KB)$ intra-order edges and $O(|\mathcal{I}|B)$ edges between order paths. Thus, the reduction can be finished in $O(2KB + |\mathcal{I}|B)$, which is upper bounded by $O(2KU + |\mathcal{I}|U)$. Besides, given a set of dispatching trees, it is easy to check whether all the order paths are covered exactly once, which guarantees that the problem is NP. Thus, the decision version of 2-DTC is NP-complete. □

As 2-DTC's decision version is already NP-complete, we naturally have Corollary 1 on the NP-hardness of 2-DTC itself.

Corollary 1. *The 2-DTC problem is NP-hard.*

As $\mathcal{D}_2 \subseteq \mathcal{D}$, 2-DTC can be regarded as a special case of the DTC problem, and thus we have Corollary 2.

Corollary 2. *The DTC problem is NP-hard.*

IV. Proposed Algorithms

A. Largest-Tree-First DTC Algorithm

We present our *Largest-Tree-First DTC (LTF-DTC)* Algorithm in Algorithm 1, which solves the DTC problem with a guaranteed approximation ratio. Our design philosophy is to efficiently search in each iteration the *largest dispatching tree*, which is the dispatching tree that covers the most order paths in the order graph, until the entire graph is covered.

The input of the LTF-DTC algorithm is the order graph $\mathcal{G}$. Algorithm 1 starts with initializing an empty DT cover $\mathcal{F}$ (line 1). The main loop (line 2-7) is the process of iteratively searching the largest dispatching tree, as long as the remaining order graph is non-empty. Specifically, in each iteration, the algorithm searches the largest dispatching tree τ in the current order graph (line 3), and includes τ into the DT cover $\mathcal{F}$ (line 4). Next, it lets $\overline{\mathcal{V}}$ be the set of vertices covered by tree τ (line 5), removes $\overline{\mathcal{V}}$ from the vertex set $\mathcal{V}$ (line 6), and removes the edges incident to these vertices from the edge set $\mathcal{E}$ (line 7). Finally, the algorithm returns the DT cover $\mathcal{F}$ (line 8).

Algorithm 1: LTF-DTC Algorithm

```
  Input: order graph $\mathcal{G}$;
  Output: DT cover $\mathcal{F}$;
  // Initialization.
1 $\mathcal{F} \leftarrow \emptyset$;
  // Iteratively search the largest DT.
2 while $\mathcal{G}$ is non-empty do
3     find the largest feasible dispatching tree $\tau$ of $\mathcal{G}$ ;
4     $\mathcal{F} \leftarrow \mathcal{F} \cup \{\tau\}$;
      // Delete all vertices covered by $\tau$.
5     $\overline{\mathcal{V}} \leftarrow \{v_{i,k} : v_{i,k} \in \tau\}$;
6     $\mathcal{V} \leftarrow \mathcal{V} \setminus \overline{\mathcal{V}}$;
      // Delete all edges incident to $\tau$.
7     $\mathcal{E} \leftarrow \mathcal{E} \setminus \{(v_{i,k}, v_{j,l}) \in \mathcal{E} | v_{i,k} \in \overline{\mathcal{V}}$ or $v_{j,l} \in \overline{\mathcal{V}}\}$;
8 return $\mathcal{F}$;
```

Note that it is highly non-trivial to find the largest dispatching tree in each iteration, and we will elaborate on addressing this issue in Section IV-B. Next, we provide theoretical analysis of the approximation ratio of Algorithm 1 in Theorem 2.

Theorem 2. *Algorithm 1 has a $1 + \ln\lambda$ approximation ratio, where λ denotes the size of the largest tree in the order graph.*

Proof. We start the proof of this theorem by introducing some extra notations. We use OPT to denote the optimal solution for DT cover, OPT_i to denote the optimal solution after the ith iteration of Algorithm 1, and f_i to denote the number of orders that have been covered after i iterations. Thus, we have that $\text{OPT}_i \le N - f_i$, because the optimal number of trees can never exceed the remaining number of orders.

Moreover, the optimal solutions in two consecutive iterations do not increase, i.e., $\text{OPT}_i \le \text{OPT}_{i-1}$. Because a dispatching tree represents a dispatching plan for one vehicle, if we take some orders off from a tree, the remaining ones can still be served by one vehicle. Therefore, the optimal solution of the last iteration after we cut off all the orders covered by a tree selected by Algorithm 1 is still a feasible solution. Naturally, we have $\text{OPT}_i \le \text{OPT}_{i-1} \le \cdots \le \text{OPT}$.

Thus far, we have two constraints on OPT_i. The right-hand-side of the first constraint, i.e., $N - f_i$, decreases monotonically with the increase of f_i. At first, $N - f_i \ge \text{OPT}$. However, as the algorithm proceeds, $N - f_i$ becomes closer to OPT. Such observation indicates that as long as $\text{OPT} \ne 1$, there exists a threshold k, such that $f_i \le N - \text{OPT}, \forall i < k$.

For each iteration $i < k$, we have that

$$f_i - f_{i-1} \ge \frac{N - f_{i-1}}{\text{OPT}_{i-1}} \ge \frac{N - f_{i-1}}{\text{OPT}},$$

which can be transformed into

$$f_i \ge \frac{N}{\text{OPT}} + \frac{\text{OPT} - 1}{\text{OPT}} f_{i-1}.$$

Then, by iteration, we have

$$f_i \ge \frac{N}{\text{OPT}}\Big(1 + \frac{\text{OPT} - 1}{\text{OPT}} + \cdots + \big(\frac{\text{OPT} - 1}{\text{OPT}}\big)^{i-1}\Big) = N\Big(1 - \big(\frac{\text{OPT} - 1}{\text{OPT}}\big)^{i}\Big).$$

As $f_i \le N - \text{OPT}$, the above inequality becomes

$$\text{OPT} \le N\Big(1 - \frac{1}{\text{OPT}}\Big)^{i} \le N \exp\Big(-\frac{i}{\text{OPT}}\Big).$$

Thus, the threshold k satisfies $k \le \text{OPT} \ln\Big(\frac{N}{\text{OPT}}\Big)$. For the second stage, there are no greater than OPT orders left, and the total number of steps g satisfies

$$g \le k + \text{OPT} \le \text{OPT} + \text{OPT} \ln\Big(\frac{N}{\text{OPT}}\Big).$$

Then, ratio of $\frac{g}{\text{OPT}}$ satisfies that

$$\frac{g}{\text{OPT}} \le 1 + \ln\Big(\frac{N}{\text{OPT}}\Big) \le 1 + \ln\lambda,$$

where λ in the size of the largest dispatching tree. □

In real-world MoD systems, the value of λ is typically small enough so that the approximation ratio given in Theorem 2 is meaningful in practice. In fact, there even exist two special cases given in the following Corollary 3, where the LTF-DTC Algorithm can provide the exact optimal solution.

Corollary 3. *The LTF-DTC Algorithm returns the optimal solution in either of the following special cases.*

- ***Case 1:*** *There exist no shareable or successional orders.*
- ***Case 2:*** *The optimal solution is 1, i.e.,* $\text{OPT} = 1$.

Proof. In Case 1, as no shareable or successional orders exist, i.e., $\lambda = 1$, Algorithm 1 returns the exact optimal solution. Then, in Case 2, as OPT$= 1$, the dispatching tree that covers all the order paths coincides with the largest dispatching tree, which will be returned in the first iteration of Algorithm 1. □

B. DP Largest Dispatching Tree Algorithm

In this section, we propose in Algorithm 2 the *dynamic programming-based Largest dispatching Tree (DP-LDT)* algorithm to find the largest dispatching tree in Algorithm 1.

In this paper, we focus on the case with $C = 3$, and our rationale for such choice is two-fold. On one hand, although a typical vehicle[2] in a MoD system has over 3 passenger seats, it usually takes no more than 3 passengers simultaneously in practice. On the other hand, to avoid passenger experience degradation from in-vehicle overcrowdedness, it is desirable that a vehicle takes no more than 3 passengers at the same time. Thus, it is reasonable and practical enough to choose $C = 3$. Our algorithms and analyses can be easily extended to $C \geq 4$ by considering additional cases.

The input of the DP-LDT algorithm is the order graph $\mathcal{G}$. It starts with associating each vertex $v_{i,j}$ with l_i (i.e., the length of the order o_i's trajectory) values, denoted as $f^r_{i,j}$ with $r \in \{0, 1, \cdots, l_i - 1\}$, and initializing all the values as 0 (line 1-3). Note that $f^r_{i,j}$ represents the largest number of orders one vehicle can serve from time slot $t_{i,j}$ to the end of the planning horizon, if order o_i shares the vehicle with another order whose starting time is earlier than o_i during the first r time slots. Clearly, when o_i is the dominating order, r can not exceed $l_i - 1$, and that is the reason why $r \in \{0, 1, \cdots, l_i - 1\}$.

Algorithm 2: DP-LDT Algorithm

```
   Input: order graph $\mathcal{G}$;
   Output: the largest dispatching tree $\tau$;
   // Initialization.
1  foreach vertex $v_{i,j}$ in $\mathcal{G}$ do
2      foreach $r \in \{0, 1, \cdots, l_i - 1\}$ do
3          $f^r_{i,j} \leftarrow 0$;
   // Update the values of each vertex.
4  foreach $n \in \{T, T-1, \cdots, 1\}$ do
5      foreach $v_{i,j}$ s.t. $t_{i,j} = n$ do
6          foreach $r \in \{0, 1, \cdots, l_i - 1\}$ do
7              if $n_i = 3$ or $\xi_i = 0$ then
8                  update $f^r_{i,j}$ according to Theorem 3;
9              else if $n_i = 2$ then
10                 update $f^r_{i,j}$ according to Theorem 4;
11             else
12                 update $f^r_{i,j}$ according to Theorem 5;
   // Find the root node.
13 $i^* \leftarrow \operatorname{argmax}_{i:v_{i,1} \in \mathcal{G}} \{f^0_{i,1}\}$;
14 set the root of the largest dispatching tree $\tau$ as $v_{i^*,1}$;
15 return $\tau$ according to the dynamic programming records;
```

Then, the algorithm utilizes dynamic programming to calculate each value $f^r_{i,j}$ in the following three cases (line 4-12). If order o_i has 3 passengers or is a non-ridesharing order (i.e., $n_i = 3$ or $\xi_i = 0$), each $f^r_{i,j}$ is updated according to Theorem 3 (line 7-8). If order o_i has 2 passengers (i.e., $n_i = 2$), we update each $f^r_{i,j}$ according to Theorem 4 (line 9-10). Otherwise, $f^r_{i,j}$ is updated according to Theorem 5. Note that we not only record each $f^r_{i,j}$'s value, but also how to achieve them. After calculating all $f^r_{i,j}$'s, we set the vertex with the largest $f^0_{i,1}$ as the root of the largest tree τ (line 13-14). Finally, we return the largest dispatching tree τ according to the dynamic programming records (line 15).

[2]As minivans and buses are still relatively rare in today's MoD systems, we do not consider them in our model.

Next, we discuss our method of calculating the value $f^r_{i,j}$ in the aforementioned 3 cases in the following Theorems 3, 4, and 5. For simplicity of presenting these theorems, we first define two functions in the following Equations (3) and (4).

$$h\big(\mathcal{S}, f^z_{x,y}\big) = \begin{cases} 1 + \max\limits_{s:o_s \in \mathcal{S}} f^z_{x,y}, & \text{if } \mathcal{S} \neq \emptyset, \\ 0, & \text{otherwise,} \end{cases} \tag{3}$$

and

$$w\big(\mathcal{S}, f^z_{x,y}\big) = \begin{cases} 2 + \max\limits_{s,t:o_s,o_t \in \mathcal{S}} f^z_{x,y}, & \text{if } |\mathcal{S}| \geq 2, \\ 0, & \text{otherwise.} \end{cases} \tag{4}$$

In both equations, $\mathcal{S}$ is a set of orders, and x, y, and z are indices that depend on i, j, r, s, and t. Then, for order o_i, we use $\mathcal{S}^u_i = \{o_s | (v_{i,l_i}, v_{s,1}) \in \mathcal{E}, t_{i,l_i} < t_{s,1}\}$ to denote the set of orders that are successional to o_i. At time slot $t_{i,j}$, all of the orders shareable with o_i can be divided into four categories by the number of passengers and whether it changes the dominating order (i.e., it finishes later than o_i). Therefore, we adopt $\mathcal{S}^\alpha_{i,j,n} = \{o_s | (v_{i,j}, v_{s,1}) \in \mathcal{E}, t_{s,l_s} \leq t_{i,l_i}, n_j = n\}$ to represent the set of orders with n passengers that can share the vehicle with o_i and do not change the dominating order, and if the dominating order changes, the set is denoted as $\mathcal{S}^\gamma_{i,j,n} = \{o_s | (v_{i,j}, v_{s,1}) \in \mathcal{E}, t_{i,l_i} > t_{s,l_s}, n_j = n\}$. Clearly, $n \in \{1, 2\}$, because orders with 3 passengers are not shareable with any other order, due to the capacity constraint.

Next, we show how to calculate $f^r_{i,j}$ for orders with $n_i = 3$ or $\xi_i = 0$ in Theorem 3. In this case, we only need to consider the orders successional to o_i. Since we are searching the largest dispatching tree, the vehicle should be dispatched to order o_s successional to o_i with the largest $f^0_{s,1}$.

Theorem 3. *For each order o_i with $n_i = 3$ or $\xi_i = 0$,*

$$f^r_{i,j} = \begin{cases} h\Big(\mathcal{S}^u_i, f^0_{s,1}\Big), & \text{if } j = 1 \text{ and } r = 0, \\ 0, & \text{otherwise.} \end{cases}$$

Then, we show how to calculate values for orders with two passengers in Theorem 4. The proof is given in Appendix A.

Theorem 4. *For each order o_i with $n_i = 2$ and $\xi_i = 1$,*

- if $r < j < l_i$,

$$f^r_{i,j} = \max\Big\{f^r_{i,j+1}, h\Big(\mathcal{S}^\alpha_{i,j,1}, f^{j+l_s-1}_{i,j+l_s}\Big), h\Big(\mathcal{S}^\gamma_{i,j,1}, f^{l_i-j+1}_{s,l_i-j+2}\Big)\Big\},$$

- if $j = l_i$,

$$f^r_{i,j} = \max\Big\{h\Big(\mathcal{S}^u_i, f^0_{s,1}\Big), h\Big(\mathcal{S}^\alpha_{i,j,1}, f^{j+l_s-1}_{i,j+l_s}\Big), h\Big(\mathcal{S}^\gamma_{i,j,1}, f^{l_i-j+1}_{s,l_i-j+2}\Big)\Big\},$$

- and otherwise $f^r_{i,j} = 0$.

Next, we discuss the calculations of $f^r_{i,j}$'s for orders with $n_i = 1$ and $\xi_i = 1$ in the following Theorem 5. Please refer to Appendix B for the proof.

Theorem 5. *For each order o_i with $n_i = 1$ and $\xi_i = 1$,*

- if $j \leq r$,

$$f^r_{i,j} = \max\Big\{f^r_{i,j+1}, h\Big(\mathcal{S}^\alpha_{i,j,1}, f^{\max\{j+l_s-1,r\}}_{i,\min\{j+l_s,r+1\}}\Big), h\Big(\mathcal{S}^\gamma_{i,j,1}, f^{l_i-j+1}_{s,r-j+2}\Big)\Big\},$$

- if $r < j < l_i$,

$$f^r_{i,j} = \max\Big\{f^r_{i,j+1}, w\Big(\mathcal{S}^\alpha_{i,j,1}, f^{j+\max\{l_s,l_t\}-1}_{i,j+\min\{l_s,l_t\}}\Big), h\Big(\mathcal{S}^\alpha_{i,j,1}, f^{j+l_s-1}_{i,j+1}\Big),$$
$$h\Big(\mathcal{S}^\alpha_{i,j,2}, f^{j+l_s-1}_{i,j+l_s}\Big), h\Big(\mathcal{S}^\gamma_{i,j,1}, f^{l_i-j+1}_{s,1}\Big), h\Big(\mathcal{S}^\gamma_{i,j,2}, f^{l_i-j+1}_{s,l_i-j+2}\Big)\Big\},$$

- otherwise,

$$f^r_{i,j} = \max\Big\{h\Big(\mathcal{S}^u_p, f^0_{s,1}\Big), w\Big(\mathcal{S}^\alpha_{i,j,1}, f^{j+\max\{l_s,l_t\}-1}_{i,j+\min\{l_s,l_t\}}\Big), h\Big(\mathcal{S}^\alpha_{i,j,1}, f^{j+l_s-1}_{i,j+1}\Big), h\Big(\mathcal{S}^\alpha_{i,j,2}, f^{j+l_s-1}_{i,j+l_s}\Big), h\Big(\mathcal{S}^\gamma_{i,j,1}, f^{l_i-j+1}_{s,1}\Big), h\Big(\mathcal{S}^\gamma_{i,j,2}, f^{l_i-j+1}_{s,l_i-j+2}\Big)\Big\}.$$

Given an order graph $\mathcal{G}$, we use D to denote the vertices' largest out-degree, N to denote the number of orders, and K to denote the length of the longest order path. Next, we provide the computational complexity of Algorithm 2 in Theorem 6.

Theorem 6. *The computational complexity of Algorithm 2 is* $O\big(D^2K|\mathcal{V}|\big)$.

As Algorithm 1 has no more than N iterations, it thus has the computational complexity given in Corollary 4.

Corollary 4. *The computational complexity of Algorithm 1 is* $O\big(D^2KN|\mathcal{V}|\big)$.

V. Online DTC Problem

Combining Algorithms 1 and 2, we could well address the DTC problem in the offline setting, where order information is *a priori* known. However, in real-world MoD systems, orders typically *arrive in an online manner*, and the platform usually has limited information on future orders.

In this section, we augment the LTF-DTC algorithm for the online setting, where the platform makes dispatching decisions at the beginnings of a series of equal-length time windows, denoted as $\mathcal{W} = \{1, 2, \cdots, W\}$, and it only has information of orders in the current and past time windows. Note that the length of a time window typically equals to multiple time slots.

In such online setting, the DTC problem still aims to minimize the number of vehicles needed over the entire planning horizon. We formally describe the *online LTF-DTC (o-LTF-DTC)* algorithm in the following Algorithm 3.

Algorithm 3: The Online LTF-DTC Algorithm

```
Input: order graph $\mathcal{G}_i$, previous DT cover $\mathcal{F}_{i-1}$;
Output: new DT cover $\mathcal{F}_i$;
1  foreach $\tau_h \in \mathcal{F}_{i-1}$ do
2      $b_h \leftarrow$ the largest set of order paths in $\mathcal{G}_i$ that can be
         added to $\tau_h$ together;
3      $\tau_h \leftarrow b_h \cup \tau_h$;
4      $\overline{\mathcal{V}}_h \leftarrow$ the set of vertices of the order paths in $b_h$;
5      $\mathcal{V}_i \leftarrow \mathcal{V}_i \setminus \overline{\mathcal{V}}_h$;
6      $\mathcal{E}_i \leftarrow \mathcal{E}_i \setminus \big\{(v_{j,k}, v_{p,q}) \in \mathcal{E}_i | v_{j,k} \in \overline{\mathcal{V}}_h$ or $v_{p,q} \in \overline{\mathcal{V}}_h\big\}$;
7      if $\mathcal{G}_i$ is empty then
8          $\widehat{\mathcal{F}} \leftarrow \emptyset$;
9          break;
10 if $\mathcal{G}_i$ is not empty then
11     $\widehat{\mathcal{F}} \leftarrow$the output of Algorithm 1 over the graph $\mathcal{G}_i$.
12 $\mathcal{F}_i \leftarrow \mathcal{F}_{i-1} \cup \widehat{\mathcal{F}}$;
13 return $\mathcal{F}_i$;
```

The o-LTF-DTC algorithm runs at the beginning of each time window $w_i \in \mathcal{W}$. Algorithm 3 takes as input the order graph $\mathcal{G}_i$ consisting of the orders which start in time window w_i and the DT cover $\mathcal{F}_{i-1} = \{\tau_1, \tau_2, \cdots, \tau_n\}$ of the previous time windows. For each dispatching tree τ_h in $\mathcal{F}_{i-1}$, the algorithm searches the largest set of order paths in $\mathcal{G}_i$ that can be added to τ_h together (line 2) using a dynamic programming-based method similar to Algorithm 2 with minor adaptations. Then, it includes b_h into τ_h, deletes all the vertices in b_h and the edges incident to them from $\mathcal{G}_i$ (line 3-6). If $\mathcal{G}_i$ becomes empty in any iteration, the loop terminates immediately, and the new dispatching tree set $\widehat{\mathcal{F}}$ is set as empty, as no extra vehicles are needed (line 7-9). Otherwise, the loop terminates after all dispatching trees in $\mathcal{F}_{i-1}$ have been considered. Then, if there still exist order paths in $\mathcal{G}_i$, the algorithm decides the new vehicles to cover them by running Algorithm 1 (line 10-11). Finally, we add the new dispatching trees $\widehat{\mathcal{F}}$ to the $\mathcal{F}_{i-1}$ to get the new DT cover $\mathcal{F}_i$, and return $\mathcal{F}_i$ (line 12-13).

The idea behind Algorithm 3 is that we first dispatch as many orders as possible to existing vehicles. However, if there exist some orders unable to be served by them, we dispatch new vehicles. For the city-scale computation, Algorithm 3 can be easily run in a distributed way, where the whole city can be divided into several small regions using some machine-learning based clustering algorithms, such as hierarchical clustering [16–18] and each region runs the algorithm itself. Therefore, Algorithm 3 can be readily adapted to cope with the scenarios with stricter constraints on computational efficiency.

VI. Experiments

A. Experiment Setups

1) Dataset: We conduct experiments with a city-scale taxi order dataset, containing over 21 million orders in Shenzhen, China, from June 1st to 30th, 2017. Each piece of data contains *time*, *car ID*, *order ID*, and *current longitude and latitude*. Although the dataset lacks the passengers' willingness to share the vehicle and the number of passengers of each order, it truly reflects the urban mobility demands. In our experiments, we randomly assign values to each order o_i's n_i and ξ_i.

2) Graph Construction: With the GPS coordinates, we divide Shenzhen into $1\text{km} \times 1\text{km}$ grids. The length of each time slot is set to be 1 minute. With such settings, we construct order graphs for our experiments, where each order is shareable with at least one order, whereas the orders in the sparse order graphs are chosen randomly from our dataset.

3) Baseline Algorithm: We introduce a competitive yet intuitive baseline *First-Shareable-Order-First (FSOF)* algorithm, which works as follows. (i) If a vehicle is serving orders, it will immediately pick up any order that is shareable with the ones being served. Otherwise, it will pick up the latest servable order received by the platform. If there are multiple servable orders at the same time, the platform will assign one of them to the vehicle uniformly at random. (ii) Repeatedly add vehicles by policy (i) until all the orders are served. Note that the online and offline versions of FSOF are exactly the same, because the decisions for each vehicle do not rely on future information.

As the DTC problem is NP-hard, it is computationally expensive to find its optimal solution. For example, even if an order graph has only 10 orders, the size of the feasible dispatching tree set $\mathcal{D}$ is in the order of 2^{10}, and thus the complexity to search for the optimal solution is in the order of 2^{1024}. It indicates that finding the optimal solution even for a small-scale problem is computationally intractable, let alone in the city scale. Therefore, we omit the comparison between our LPF-DTC algorithm and the optimal solution. Moreover, we are not able to compare with algorithms in past literatures, as no existing works study exactly the same problem as ours.

B. Experiment Results

1) Evaluation for the 3-DTC Problem: We first consider a special case of the DTC problem, namely the *3-DTC problem* where each order o_i has only one passenger (i.e., $n_i = 1$).

In Figure 4, we compare LTF-DTC with FSOF in an order graph derived from our dataset, respectively. We can observe that the number of vehicles needed by LTF-DTC increases only very slightly after the number of orders reaches 4000. In contrast, the result of FSOF increases almost linearly. When the number of orders reaches 9000, the number of vehicles needed by LTF-DTC is only 21% of that of FSOF.

In Figure 6, we vary the percentage of ridesharing orders from 0% to 100%, and the numbers given by both LTF-DTC and FSOF decrease as such percentage increases. Although the gap between them is not obvious at first, LTF-DTC still has a huge improvement compared with FSOF in most cases. Specifically, when the percentage of ridesharing orders reaches 100%, LTF-DTC only uses 52% vehicles needed by FSOF.

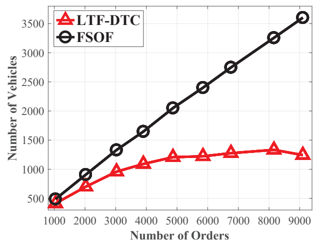

Fig. 4: 3-DTC.

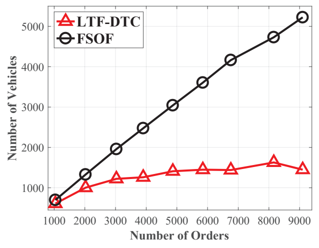

Fig. 5: General DTC (dense).

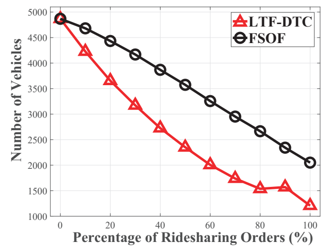

Fig. 6: 3-DTC (vary the percentage of ridesharing orders).

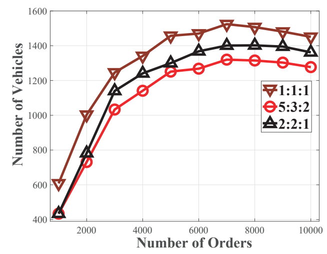

Fig. 7: General DTC (vary the distribution of passenger number).

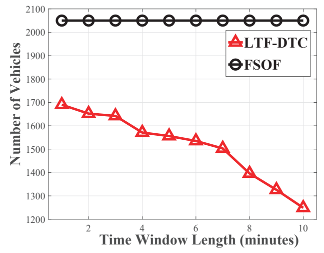

Fig. 8: Online DTC (5,000 orders).

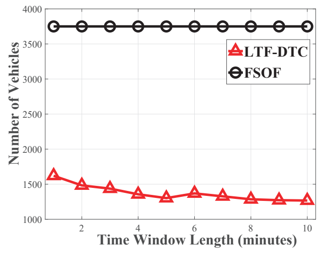

Fig. 9: Online DTC (10,000 orders).

2) Evaluation for the General DTC Problem: We evaluate LTF-DTC for the general DTC problem, where we assign each order's passenger number which ranges from 1 to 3 randomly.

Figure 5 compare LTF-DTC with FSOF with uniformly distributed passenger numbers. Figure 5 has a similar trend as Figure 4, and shows that LTF-DTC needs about one fifth vehicles needed by FSOF, when the overall order number reaches 9,000.

We then vary the distribution each order's passenger numbers in the dense order graph setting, and show the results in Figure 7, where legend $x : y : z$ denotes the ratio of orders with 1, 2, and 3 passengers, respectively. This figure shows a rough trend of growth in the vehicle numbers with higher percentage of orders with 2 or 3 passengers. Moreover, the difference of the results under different distributions is slight, which shows the robustness of our LTF-DTC algorithm.

3) Evaluation for the Online DTC Problem: We now evaluate our online LTF-DTC algorithm with 5,000 and 10,000 orders that are densely sharable from the dataset.

Figures 8 and 9 show the results with the time window varying from 1 to 10 minutes, indicating that a longer time window is more beneficial to online LTF-DTC's performance. However, even in the worst case where a time window lasts 1 minute, our algorithm still outperforms FSOF by 17.6% in the setting with 5,000 orders. When a time window equals to 10 minutes, our algorithm outperforms FSOF by at least 39.1%. We can observe a wider performance gap between online LTF-DTC and FSOF when the order number equals to 10,000, where LTF-DTC outperforms FSOF stably by at least 54.1%.

VII. Related Work

Optimizing MoD systems [13,19–22] has always been an imperative research topic since its introduction by companies like Uber and DiDi. Also, ridesharing is a voguish research topic [3,4,6,23,24]. Much efforts have been made to lay basis for optimizing ridesharing systems [25–28].

The economic aspect of MoD systems has been studied by a wide variety of researches [1,8,29–35]. Most of them focus on pricing [1,29–31] to maximize the profits of MoD systems. [31] provides the economics insights of car-pooling with self-driving cars. Apart from the pure economic aspect, our work focuses on minimizing the number of vehicles while satisfying all the mobility demands, which brings benefits for the whole society as well as cuts budget for MoD service providers.

Except for the economic aspect, optimization for routing [3,36] and order dispatch [6–9] in a ridesharing system is another widely investigated research topic. Among the algorithms for order dispatching, various settings, such as on-demand order dispatching [6,7], order dispatching with auctions [8], and dynamic dispatching [9], have been studied. Joint optimization of order dispatch and routing has been addressed by a probabilistic framework [4]. With the great advances in machine learning, deep learning and reinforcement learning approaches are proposed to solve routing [37] and order dispatching [2,38–40] problems. However, the above existing prior works take different objectives form ours which aims at using the minimal number of vehicles to satisfy heterogeneous mobility demands. Besides, the machine-learning based methods usually do not come with complete interpretability of their results, whereas our work provides rigorous theoretical analysis of the performance bounds of our algorithms.

Note that a previous work [5] has addressed the minimum fleet problem for traditional ridesharing-free taxi systems. However, the minimum fleet problem for ridesharing-aware MoD systems with heterogeneous mobility demands investigated in this paper is theoretically fundamentally different from the minimum taxi fleet problem, and thus makes the methods proposed in [5] inapplicable in our problem setting.

VIII. Conclusion

In this paper, we address the fundamental problem in ridesharing-aware MoD systems of how to serve urban residents' heterogeneous transportation demands with as few vehicles as possible. We use graph-theoretic methods to construct an efficient and realistic model considering various real-world factors and formulate our problem into a tree cover problem, different from the traditional coverage problems. Theoretically, we prove the tree cover problem is NP-hard. In order to solve it, we propose a dynamic programming-based polynomial-time algorithm with a guaranteed approximation ratio. Furthermore, we address the online fleet minimization problem, where orders arrive sequentially in an online manner. Experimentally, extensive experiments both for the online and offline problem on a city-scale dataset from Shenzhen containing 21 million passenger orders from July 1st to 30th, 2017, validate the effectiveness of our algorithms.

Acknowledgement

This work was supported in part by National Key R&D Program of China 2018AAA0101200, in part by NSFC China (No. 61902244, U20A20181, 61960206002, 61822206, 62020106005, 61829201, 62041205, 61532012), in part by Shanghai Municipal Science and Technology Commission Grant 19YF1424600.

Appendix A

Proof of Theorem 4. If $n_i = 2$ and $\xi_i = 1$, order o_i can only share a vehicle with the orders that have one passenger. If $j \le r$, there must have been three passengers in the vehicle that is serving order o_i already. Thus, we have $f_{i,j}^r = 0$.

$j > r$ corresponds to the scenario where any order sharing the vehicle with o_i at the first r time slots has been dropped off. Then, due to the capacity constraints, it can only pick up an order o_q that is either a ridesharing order with one passenger, or a successional order. Thus, we calculate $f_{i,j}^r$ in four cases.

- **Case 1** ($o_q \in \mathcal{S}_{i,j,1}^{\alpha}$): After being picked up, o_q will share the vehicle with o_i in the next l_q time slots, so the vehicle cannot serve any new order until $t_{i,j+l_q}$. Thus, we have
$$f_{i,j}^r = 1 + \max_{s:o_s \in \mathcal{S}_{i,j,1}^{\alpha}} f_{i,j+l_s}^{j+l_s-1},$$
and $q = \operatorname{argmax}_{s:o_s \in \mathcal{S}_{i,j,1}^{\alpha}} f_{i,j+l_s}^{j+l_s-1}$.
- **Case 2** ($o_q \in \mathcal{S}_{i,j,1}^{\gamma}$): After o_q is picked up, the dominating order changes from o_i to o_q. Only after dropping off o_i at t_{i,l_i-j+2} can the vehicle pick up new orders. Thus, we have
$$f_{i,j}^r = 1 + \max_{s:o_s \in \mathcal{S}_{i,j,1}^{\gamma}} f_{s,l_i-j+2}^{l_i-j+1},$$
and $q = \operatorname{argmax}_{s:o_s \in \mathcal{S}_{i,j,1}^{\gamma}} f_{s,l_i-j+2}^{l_i-j+1}$.
- **Case 3** ($j < l_i$, no order pick-up): In this case, $f_{i,j}^r = f_{i,j+1}^r$.
- **Case 4** ($j = l_i$, $o_q \in \mathcal{S}_i^u$): In this case, the vehicle is assigned an order successional to o_i. Then, the value of $f_{i,j}^r$ is
$$f_{i,j}^r = 1 + \max_{s:o_s \in S_i^u} f_{s,1}^0,$$
and $q = \operatorname{argmax}_{s:o_s \in S_i^u} f_{s,1}^0$.

Since we are searching the largest dispatching tree, for $r < j < l_i$, $f_{i,j}^r$ is chosen to be the largest among Cases 1-3, i.e.,
$$f_{i,j}^r = \max\left\{ f_{i,j+1}^r, h\Big(\mathcal{S}_{i,j,1}^{\alpha}, f_{i,j+l_s}^{j+l_s-1}\Big), h\Big(\mathcal{S}_{i,j,1}^{\gamma}, f_{s,l_i-j+2}^{l_i-j+1}\Big) \right\}.$$
When $j = l_i$, Case 4 replaces Case 3. □

Appendix B

Proof of Theorem 5. If $j \le r$, order o_i is sharing a vehicle with some other order o_p which will not get off until time slot $t_{i,r}$. Thus, if the vehicle can still pick up a new order o_q, both o_p and o_q can have only one passenger. Then, we calculate $f_{i,j}^r$ respectively in the 3 cases: $o_q \in \mathcal{S}_{i,j,1}^{\alpha}$, $o_q \in \mathcal{S}_{i,j,1}^{\gamma}$ and no order pick-up. As it is very similar to the proof of Theorem 4, we directly have if $j \le r$,
$$f_{i,j}^r = \max\left\{ f_{i,j+1}^r, h\Big(\mathcal{S}_{i,j,1}^{\alpha}, f_{i,\min\{j+l_s,r+1\}}^{\max\{j+l_s-1,r\}}\Big), h\Big(\mathcal{S}_{i,j,1}^{\gamma}, f_{s,r-j+2}^{l_i-j+1}\Big) \right\}.$$

$j > r$ corresponds to the scenario where there is only o_i in the vehicle. Then, the vehicle can pick up at most two orders o_q and o_g. Thus, $f_{i,j}^r$ is calculated in the following 7 cases.

- **Case l1** ($o_q, o_g \in \mathcal{S}_{i,j,1}^{\alpha}$): The vehicle cannot take new orders until either o_q or o_g gets off, and o_i shares the vehicle with one of them until time slot $t_{i,j+\max\{l_q,l_g\}-1}$. Thus, we have
$$f_{i,j}^r = 2 + \max_{(s,t):o_s,o_t \in \mathcal{S}_{i,j,1}^{\alpha}} f_{i,j+\min\{l_s,l_t\}}^{j+\max\{l_s,l_t\}-1},$$
and $(q,g) = \operatorname{argmax}_{(s,t):o_s,o_t \in \mathcal{S}_{i,j,1}^{\alpha}} f_{i,j+\min\{l_s,l_t\}}^{j+\max\{l_s,l_t\}-1}$.
- **Case l2** ($o_q \in \mathcal{S}_{i,j,1}^{\alpha}$): In this case, after picking up o_q, the vehicle can still pick up orders in the next time slot, and o_i will share the vehicle until time $t_{i,j+l_q+1}$. Thus, we have
$$f_{i,j}^r = 1 + \max_{s:o_s \in \mathcal{S}_{i,j,1}^{\alpha}} f_{i,j+1}^{j+l_s-1},$$
and $q = \operatorname{argmax}_{s:o_s \in \mathcal{S}_{i,j,1}^{\alpha}} f_{i,j+1}^{j+l_s-1}$.
- **Case l3** ($o_q \in \mathcal{S}_{i,j,2}^{\alpha}$): In this case, the vehicle cannot pick up any new order until the time slot $t_{i,j+l_q}$. Thus, we have
$$f_{i,j}^r = 1 + \max_{s:o_s \in \mathcal{S}_{i,j,2}^{\alpha}} f_{i,j+l_s}^{j+l_s-1},$$
and $q = \operatorname{argmax}_{s:o_s \in \mathcal{S}_{i,j,2}^{\alpha}} f_{i,j+l_s}^{j+l_s-1}$.
- **Case l4** ($o_q \in \mathcal{S}_{i,j,1}^{\gamma}$): o_q becomes the dominating order, and it shares its first $l_i - j + 1$ time slots with o_i. Thus, we have
$$f_{i,j}^r = 1 + \max_{s:o_s \in \mathcal{S}_{i,j,1}^{\gamma}} f_{s,1}^{l_i-j+1},$$
and $q = \operatorname{argmax}_{s:o_s \in \mathcal{S}_{i,j,1}^{\gamma}} f_{s,1}^{l_i-j+1}$.
- **Case l5** ($o_q \in \mathcal{S}_{i,j,2}^{\gamma}$): Different from Case l4, the vehicle can only pick up new orders from t_{i,l_i-j+2}. Thus, we have
$$f_{i,j}^r = 1 + \max_{s:o_s \in \mathcal{S}_{i,j,2}^{\gamma}} f_{s,l_i-j+2}^{l_i-j+1},$$
and $q = \operatorname{argmax}_{s:o_s \in \mathcal{S}_{i,j,2}^{\gamma}} f_{s,l_i-j+2}^{l_i-j+1}$.
- **Case l6** ($j < l_i$, no order pickup): In this case, $f_{i,j}^r = f_{i,j+1}^r$.
- **Case l7** ($j = l_s$, $o_q \in \mathcal{S}_i^u$): In this case, the vehicle is assigned an order successional to o_i. Therefore, we have
$$f_{i,j}^r = 1 + \max_{s:o_s \in \mathcal{S}_i^u} f_{s,1}^0,$$
and $q = \operatorname{argmax}_{s:o_s \in \mathcal{S}_i^u} f_{s,1}^0$.

For $r < j < l_i$, we update $f_{i,j}^r$ by the maximum value over Case l1-l6. Thus, we have
$$f_{i,j}^r = \max\Big\{ f_{i,j+1}^r, w\Big(\mathcal{S}_{i,j,1}^{\alpha}, f_{i,j+\min\{l_s,l_t\}}^{j+\max\{l_s,l_t\}-1}\Big), h\Big(\mathcal{S}_{i,j,1}^{\alpha}, f_{i,j+1}^{j+l_s-1}\Big),$$
$$h\Big(\mathcal{S}_{i,j,2}^{\alpha}, f_{i,j+l_s}^{j+l_s-1}\Big), h\Big(\mathcal{S}_{i,j,1}^{\gamma}, f_{s,1}^{l_i-j+1}\Big), h\Big(\mathcal{S}_{i,j,2}^{\gamma}, f_{s,l_i-j+2}^{l_i-j+1}\Big) \Big\}.$$
Case l7 replaces Case l6, when $j = l_i$. □

References

[1] C. Courcoubetis and A. Dimakis, “Throughput and pricing of ridesharing systems,” in *INFOCOM*, 2019.

[2] T. Oda and C. Joe-Wong, “Movi: A model-free approach to dynamic fleet management,” in *INFOCOM*, 2018.

[3] Q. Lin, L. Dengt, J. Sun, and M. Chen, “Optimal demand-aware ride-sharing routing,” in *INFOCOM*, 2018.

[4] Q. Lin, W. Xu, M. Chen, and X. Lin, “A probabilistic approach for demand-aware ride-sharing optimization,” in *MOBIHOC*, 2019.

[5] M. M. Vazifeh, P. Santi, G. Resta, S. H. Strogatz, and C. Ratti, “Addressing the minimum fleet problem in on-demand urban mobility,” *Nature*, vol. 557, no. 7706, pp. 534–538, 2018.

[6] J. Alonso-Mora, S. Samaranayake, A. Wallar, E. Frazzoli, and D. Rus, “On-demand high-capacity ride-sharing via dynamic trip-vehicle assignment,” *Proceedings of the National Academy of Sciences*, 2017.

[7] Y. Liu, W. Skinner, and C. Xiang, “Globally-optimized realtime supply-demand matching in on-demand ridesharing,” in *WWW*, 2019.

[8] L. Zheng, P. Cheng, and L. Chen, “Auction-based order dispatch and pricing in ridesharing,” in *ICDE*, 2019.

[9] X. Bei and S. Zhang, “Algorithms for trip-vehicle assignment in ride-sharing,” in *AAAI*, 2018.

[10] Y. Li, R. Yu, C. Shahabi, and Y. Liu, “Diffusion convolutional recurrent neural network: Data-driven traffic forecasting,” in *ICLR*, 2018.

[11] C. Zheng, X. Fan, C. Wang, and J. Qi, “Gman: A graph multi-attention network for traffic prediction,” in *AAAI*, 2020.

[12] A. Baggag, S. Abbar, A. Sharma, T. Zanouda, A. Al-Homaid, A. Mohan, and J. Srivasatava, “Learning spatiotemporal latent factors of traffic via regularized tensor factorization: Imputing missing values and forecasting,” *IEEE Transactions on Knowledge and Data Engineering*, 2019.

[13] Y. Guan, A. M. Annaswamy, and H. E. Tseng, “A dynamic routing framework for shared mobility services,” *ACM Transactions on Cyber-Physical Systems*, 2019.

[14] P.-J. Wan, D.-Z. Du, P. Pardalos, and W. Wu, “Greedy approximations for minimum submodular cover with submodular cost,” *Comput. Optim. Appl.*, vol. 45, p. 463474, Mar. 2010.

[15] M. R. Garey and D. S. Johnson, “Computers and intractability: A guide to the theory of np-completeness,” 1978.

[16] S. C. Johnson, “Hierarchical clustering schemes,” *Psychometrika*, vol. 32, no. 3, pp. 241–254, 1967.

[17] D. Ghoshdastidar, M. Perrot, and U. von Luxburg, “Foundations of comparison-based hierarchical clustering,” in *NeurlIPS*, 2019.

[18] A. Abboud, V. Cohen-Addad, and H. Houdrougé, “Subquadratic high-dimensional hierarchical clustering,” in *NeurlIPS*, 2019.

[19] S. Jiang, L. Chen, A. Mislove, and C. Wilson, “On ridesharing competition and accessibility: Evidence from uber, lyft, and taxi,” in *WWW*, 2018.

[20] T. Séjournè, S. Samaranayake, and S. Banerjee, “The price of fragmentation in mobility-on-demand services,” *SIGMETRICS*, 2018.

[21] S. Guo, C. Chen, J. Wang, Y. Liu, K. Xu, D. Zhang, and D. M. Chiu, “A simple but quantifiable approach to dynamic price prediction in ride-on-demand services leveraging multi-source urban data,” *IMWUT*, 2018.

[22] E. Wang, R. Ding, Z. Yang, H. Jin, C. Miao, L. Su, F. Zhang, C. Qiao, and X. Wang, “Joint charging and relocation recommendation for e-taxi drivers via multi-agent mean field hierarchical reinforcement learning,” *IEEE Transactions on Mobile Computing*, 2020.

[23] T. Kumsila and S. Phithakkitnukoon, “Jerney: A peer-to-peer shared public transit on fixed routes,” in *IMWUT*, 2018.

[24] C. Liu, J. Sun, H. Jin, M. Ai, Q. Li, C. Zhang, K. Sheng, G. Wu, X. Qie, and X. Wang, “Spatio-temporal hierarchical adaptive dispatching for ridesharing systems,” in *SIGSPATIAL*, 2020.

[25] J. J. Pan, G. Li, and J. Hu, “Ridesharing: Simulator, benchmark, and evaluation,” *VLDB*, 2019.

[26] D. Zhang, T. He, F. Zhang, M. Lu, Y. Liu, H. Lee, and S. H. Son, “Carpooling service for large-scale taxicab networks,” *ACM Transactions on Sensor Networks*, 2016.

[27] F. Bistaffa, J. Rodríguez-Aguilar, J. Cerquides, and C. Blum, “A simulation tool for large-scale online ridesharing,” AAMAS, 2018.

[28] L. Chen, Y. Gao, Z. Liu, X. Xiao, C. S. Jensen, and Y. Zhu, “Ptrider: A price-and-time-aware ridesharing system,” *VLDB*, 2018.

[29] L. Chen, Q. Zhong, X. Xiao, Y. Gao, P. Jin, and C. S. Jensen, “Price-and-time-aware dynamic ridesharing,” ICDE, 2018.

[30] H. Ma, F. Fang, and D. C. Parkes, “Spatio-temporal pricing for ridesharing platforms,” *EC*, 2018.

[31] M. Ostrovsky and M. Schwarz, “Carpooling and the economics of self-driving cars,” in *EC*, 2019.

[32] Z. Fang, L. Huang, and A. Wierman, “Loyalty programs in the sharing economy: Optimality and competition,” in *MOBIHOC*, 2018.

[33] Z. Fang, L. Huang, and A. Wierman, “Prices and subsidies in the sharing economy,” in *WWW*, 2017.

[34] K. Bimpikis, O. Candogan, and D. Saban, “Spatial pricing in ride-sharing networks,” *Operations Research*, 2019.

[35] S. Guo, C. Chen, J. Wang, Y. Liu, K. Xu, Z. Yu, D. Zhang, and D. M. Chiu, “Rod-revenue: Seeking strategies analysis and revenue prediction in ride-on-demand service using multi-source urban data,” *IEEE Transactions on Mobile Computing*, vol. 19, no. 9, 2020.

[36] A. Braverman, J. G. Dai, X. Liu, and L. Ying, “Empty-car routing in ridesharing systems,” *Operations Research*, 2019.

[37] Z. T. Qin, J. Tang, and J. Ye, “Deep reinforcement learning with applications in transportation,” KDD, 2019.

[38] S. He and K. G. Shin, “Spatio-temporal capsule-based reinforcement learning for mobility-on-demand network coordination,” in *WWW*, 2019.

[39] M. Li, Z. Qin, Y. Jiao, Y. Yang, J. Wang, C. Wang, G. Wu, and J. Ye, “Efficient ridesharing order dispatching with mean field multi-agent reinforcement learning,” in *WWW*, 2019.

[40] Z. Xu, Z. Li, Q. Guan, D. Zhang, Q. Li, J. Nan, C. Liu, W. Bian, and J. Ye, “Large-scale order dispatch in on-demand ride-hailing platforms: A learning and planning approach,” in *KDD*, 2018.