2020 IEEE 36th International Conference on Data Engineering (ICDE)

Mobility-Aware Dynamic Taxi Ridesharing

Zhidan Liu*†, Zengyang Gong†, Jiangzhou Li†, Kaishun Wu*†‡
*Guangdong Laboratory of Artificial Intelligence and Digital Economy (SZ), Shenzhen University, P. R. China
†College of Computer Science and Software Engineering, Shenzhen University, P. R. China
‡PCL Research Center of Networks and Communications, Peng Cheng Laboratory, Shenzhen, P. R. China
liuzhidan@szu.edu.cn, {gongzengyang2017, lijiangzhou2018}@email.szu.edu.cn, wu@szu.edu.cn

***Abstract*—Taxi ridesharing becomes promising and attractive because of the wide availability of taxis in a city and tremendous benefits of ridesharing, *e.g.*, alleviating traffic congestion and reducing energy consumption. Existing taxi ridesharing schemes, however, are not efficient and practical, due to they simply match ride requests and taxis based on partial trip information and omit the offline passengers, who hail a taxi at roadside with no explicit requests to the system. In this paper, we consider the mobility-aware taxi ridesharing problem, and present *mT-Share* to address these limitations. *mT-Share* fully exploits the mobility information of ride requests and taxis to achieve efficient indexing of taxis/requests and better passenger-taxi matching, while still satisfying the constraints on passengers' deadlines and taxis' capacities. Specifically, *mT-Share* indexes taxis and ride requests with both geographical information and travel directions, and supports the shortest path based routing and probabilistic routing to serve both online and offline ride requests. Extensive experiments with a large real-world taxi dataset demonstrate the efficiency and effectiveness of *mT-Share*, which can response each ride request in milliseconds and with a moderate detour cost. Compared to state-of-the-art methods, *mT-Share* serves 42% and 62% more ride requests in peak and non-peak hours, respectively.**

***Index Terms*—taxi ridesharing, mobility, clustering, route planning**

I. Introduction

Ridesharing allows multiple passengers with the similar itineraries and time schedules to share a vehicle, which can significantly alleviate urban traffic congestion, reduce energy consumption, and bring win-win benefits to both passengers and drivers [21]. Due to the wide availability of taxis in a city, taxi ridesharing becomes a promising transportation mode [21], [22], [41], [42]. Different from private vehicles based ridesharing, also known as carpooling [12], [29], where ride requests are static and ridesharing routes could be planned in advance, taxi ridesharing is more complex, because both ride requests and taxis are highly dynamic [21], [22]. On one hand, passengers usually submit their requests immediately once they need a ride with no prior planning. Even worse, some passengers will not explicitly report their requests but hail a taxi at roadside. On the other hand, a taxi randomly delivers passengers in the city with no fixed route. Such dynamics cause the real-time taxi ridesharing especially challenging, where ride requests need to be timely assigned to taxis and meanwhile taxi schedule and route should be wisely updated to guarantee the quality of services [14], [41].

In the literature, some remarkable efforts have been made to design taxi ridesharing schemes [13], [21], [22], [41], [42]. They usually select a set of candidate taxis for a ride request according to the passenger's current location and the geographical distribution of all taxis, and then insert this request into a candidate taxi's schedule that can minimize certain cost, *e.g.*, minimum increase in the travel cost, while still satisfying the service requirements of other passengers already in the taxi. These schemes, however, are not efficient and practical due to the following limitations. First, most existing schemes determine candidate taxis for a ride request based on partial trip information merely, *i.e.*, a ride request's origin location and taxis' current locations, and thus will not achieve the best passenger-taxi matching. Second, existing schemes mainly consider online ride requests, while some ride requests may be *offline* and thus be invisible to the ridesharing system. According to a recent taxi service research report, the statistic on users' preferences of getting taxi services shows that 41.68% users prefer either online booking or offline hailing, and 13.71% users only accept taxi-hailing in an offline manner [3]. Therefore, the amount of offline ride requests (*i.e.*, $13.71\% \sim 55.39\%$ of users) could be very large, and an appropriate taxi ridesharing scheme is desired to well handle such demands.

In this paper, we consider a more practical problem in taxi ridesharing, namely *mobility-aware taxi ridesharing* (MTR), which exploits the known mobility information from taxis and requests and the hidden mobility patterns from historical data to match passengers with suitable taxis, so that to maximize the served ride requests while minimizing the total detour cost, subject to the constraints of taxis' capacities and passengers' deadlines. It turns to be an extremely challenging problem, not only because the high dynamics of online ride requests and taxis, but also the difficulty on predicting the offline requests.

To improve existing works and solve the MTR problem, we present *mT-Share* – a novel taxi ridesharing scheme that fully exploits both real-time and historical mobility information of taxis and ride requests. The key insight behind *mT-Share* is that the best matches of taxis and ride requests should share the similar travel directions and have geographically close origins and destinations. Therefore, *mT-Share* proposes bipartite map partitioning and mobility clustering to facilitate the indexing of taxis and ride requests from these two aspects. Based on the map partitions, two routing modes are enabled and further optimized for the passenger-taxi matching, which can improve the efficiency of taxi scheduling while simultaneously satisfying the constraints on taxi's capacity and passengers' deadlines.

In particular, by considering the weekly and daily mobility patterns of taxi demands [6], [16], a novel probabilistic routing is proposed, which allows a taxi to opportunistically encounter offline ride requests with high probability. The contributions of our work thus can be summarized as follows:

- We identify the limitations of existing taxi ridesharing schemes, and are the first, to the best of our knowledge, to consider the MTR problem, which exploits the mobility information to serve both online and offline ride requests.
- We propose a novel scheme named *mT-Share* to address the MTR problem. By incorporating the holistic mobility information of taxis and ride requests, *mT-Share* has optimized the indexing of taxis/requests and passenger-taxi matching. In particular, probabilistic routing is devised to predict offline requests by mining historical taxi data.
- We conduct extensive experiments to evaluate *mT-Share* with a large real-world taxi dataset. Experimental results show that *mT-Share* significantly outperforms the state-of-the-art methods, *e.g.*, serving 42% and 62% more ride requests in the peak and non-peak hours, respectively.

The rest of this paper is organized as follows. We review the related works in Section II. We present the problem statement in Section III. The design of *mT-Share* is elaborated in Section IV. We evaluate *mT-Share* in Section V. Finally, Section VI concludes this paper.

II. Related Work

Taxi demands and dispatching. As an important public transportation mode in the urban city, it is crucial to understand taxi demands and well dispatch taxis to balance the supplies and demands [36]. A plenty of works [31], [37], [40] have been devoted to predict taxi demands by analyzing historical taxi transactions. Different from these works that predict taxi demands for regular taxi services, we predict offline ride requests in the taxi ridesharing scenario, which is more challenging. In particular, a recent work [16] proposes a demand prediction based ridesharing routing by exploiting the mobility statistics. Our work differs from it by considering both online and offline ride requests.

In addition to the taxi demand prediction, many works [15], [40], [43], [44] study the taxi dispatching problem according to the known demands. For example, Zhang *et al*. [40] propose an order dispatching model to maximize the global matching success rate of passengers and taxis. Lin *et al*. [15] provide an efficient fleet management method that exploits multi-agent deep reinforcement learning for an explicit coordination among taxis given all taxi demands. These works dispatch a vacant taxi for each ride request, with no consideration of taxi ridesharing at all.

Carpooling and Dial-A-Ride. Carpooling often refers to recurring ridesharing, which mainly deals with routine commutes, *e.g.*, from home to workplace. Since carpooling usually involves a small size of drivers and riders, it can be optimally solved using the linear programming [4]. Potential carpooling opportunity can be discovered by mining GPS trajectories [12], and *coRide* [39] could be used to design the schedules and routes. Different from carpooling where all ride requests are known in advance, taxi ridesharing is more dynamic, since ride requests are generated on the fly and taxi routes will continuously change.

The taxi ridesharing problem can be viewed as a variant of the dial-a-ride problem (DARP) that designs the schedule and route for a set of riders, who specify their pick-up and drop-off locations in advance, between origins and destinations [9]. Existing works on DARP primarily focus on the static case [8], where all ride requests are provided in a prior. The general DARP problem is NP-hard, and little research has been carried out on the dynamic DARP [22].

Ridesharing. Ridesharing has been actively studied in recent years due to its tremendous benefits. Compared to the static carpooling, dynamic ridesharing is more realistic yet challenging, which can be modelled as a combinatorial optimization problem and has been proved to be NP-hard [5]. Therefore, various heuristics have been proposed to optimize the two stages of ridesharing, *i.e.*, candidate taxi searching [29], [30] and ridesharing routing [14], [34]. For example, Tong *et al*. [34], [35] optimize the route planning with a smart insertion for the shared mobility. In addition, other factors of ridesharing systems, such as pricing models [7], [32], request assignment [18], [33], privacy protections [11], riders' attitude on ridesharing [42], and riders' satisfaction [8], have also been explored in recent years.

Due to the wide availability of taxis in a city [6], taxi ridesharing becomes promising and has already attracted some research efforts [13], [21], [22], [41], [42]. Specifically, Zheng *et al*. present a mobile-cloud based taxi-sharing system named *T-Share* [21], [22]. Zhang *et al*. improve *T-Share* by considering the quality of service for taxi ridesharing [41]. Hou *et al*. particularly study the transfer-allowed ridesharing with the battery limited electric taxis [13]. In [42], passenger's acceptance probability on taxi ridesharing is also considered. These works, however, do not make use of mobility information for better passenger-taxi matching, and meanwhile omit the offline ride requests, who do not explicitly report their requests to the ridesharing system.

Taxi trajectory mining. The availability of massive taxi trajectory data enables various novel applications [6], [45], *e.g.*, traffic estimation [20], [17], traffic prediction [19], and collective travel planning [24]. Therefore, there exist many efforts to improve the operation efficiency of joining [25], [26] and searching [27], [28] on the trajectory data. In this paper, we make use of historical taxi trajectory data to mine mobility patterns for better taxi ridesharing.

III. Problem Statement

In this section, we introduce the notations, motivation, and the formal definition about mobility-aware taxi ridesharing.

A. Notations and Definitions

Definition 1: (**Road Network**) *A road network is denoted by a directed graph* $\mathbf{G}(\mathbf{V}, \mathbf{E})$*, where each vertex* $v \in \mathbf{V}$ *presents a geo-location (e.g., road intersection), and each edge*

$(u,v) \in \mathbf{E}$ is a road segment, which is associated with a weight $cost(u,v)$, indicating the travel cost from u to v.

The travel cost can be measured as either a travel distance or a travel time. Since they can be converted from one to another when travel speed of a vehicle is known, we thus do not differentiate them and use the travel cost consistently. In this paper, we assume stable traffic conditions as previous studies [21], [34] and thus the travel cost of an edge is constant.

For simplicity, we assume that most passengers are willing to take a taxi ridesharing, which coincides with a recent report [23]. Specifically, the passengers either explicitly report their ride requests to the system through booking Apps or implicitly join the ridesharing by hailing a shared taxi at roadside. The system will select proper taxis to serve both online and offline ride requests by analyzing current statuses of all taxis, and plan routes for the chosen taxis given certain service requirements.

Definition 2: (**Ride Request**) *A ride request is denoted by $\mathbf{r_i} =< t_{r_i}, o_{r_i}, d_{r_i}, e_{r_i} >$ with a trip origin $o_{r_i} \in \mathbf{V}$ and a trip destination $d_{r_i} \in \mathbf{V}$. This request is released at time t_{r_i} and should be completed before time e_{r_i} by delivering the passengers from o_{r_i} to d_{r_i}.*

Note that two deadlines could be adopted in the real-world applications, *i.e.*, the deadlines for pick-up and drop-off [22]. In fact, a single deadline for delivery e_{r_i} usually suffices [34], since the pick-up deadline can be easily inferred from e_{r_i} and the travel time $cost(o_{r_i}, d_{r_i})$ between o_{r_i} to d_{r_i}, *i.e.*, the pick-up deadline could be expressed as $e_{r_i} - cost(o_{r_i}, d_{r_i})$. The information of an online ride request can be known once $\mathbf{r_i}$ is submitted, while offline ride requests would be known only when they are opportunistically encountered by shared taxis. We use $\bar{\mathbf{r_i}}$ to specially present an offline ride request.

Definition 3: (**Taxi Status**) *The instantaneous status of the j-th taxi is denoted by $\mathbf{t_j} =< loc_{t_j}, \mathcal{S}_{t_j}, \mathcal{R}_{t_j} >$, where loc_{t_j} presents taxi $\mathbf{t_j}$'s current location, and $\mathcal{S}_{t_j}$ and $\mathcal{R}_{t_j}$ are taxi $\mathbf{t_j}$'s schedule and route, respectively.*

Definition 4: (**Taxi Schedule**) *A valid schedule $\mathcal{S}_{t_j} = \{s_1, s_2, \cdots, s_m\}$ is a sequence of events for a shared taxi, where each event corresponds to pick-up or drop-off the ridesharing passenger at some location, e.g., o_{r_i} or d_{r_i} of a ride request $\mathbf{r_i}$ and o_{r_i} should appear ahead of d_{r_i}.*

Definition 5: (**Taxi Route**) *A taxi route $\mathcal{R}_{t_j}$ is generated according to taxi schedule $\mathcal{S}_{t_j}$, which indicates the travel path for any two consecutive events in $\mathcal{S}_{t_j}$.*

Given ride requests that share the same taxi, a valid schedule should be made to sequentially pick-up and deliver passengers along a planned ridesharing route. The travel path is usually set as the shortest path between two event locations in previous works [7], [14], [21], [22], [34], and thus a taxi route is derived by concatenating a sequence of these shortest paths. Both $\mathcal{S}_{t_j}$ and $\mathcal{R}_{t_j}$ are continuously updated when ridesharing passengers are picked up or delivered by taxi $\mathbf{t_j}$.

The key notations are summarized in Table I.

B. Motivation

The taxi ridesharing problem can be modelled as a combinatorial optimization problem, which has been proved to be NP-hard [5]. As a result, the taxi ridesharing problem cannot be optimally solved with a polynomial time complexity. Existing works [13], [21], [22], [41], [42] thus propose various heuristics to assign ride requests to shared taxis. In general, these schemes index taxis with grids of the road network, and process each ride request $\mathbf{r_i}$ through the following two stages.

Stage 1: taxi searching. The taxis within a range γ around $\mathbf{r_i}$'s origin o_{r_i} are selected as the candidate taxis to serve $\mathbf{r_i}$.

Stage 2: ridesharing routing. The schedule of each candidate taxi $\mathbf{t_j}$ is examined by inserting the pick-up and drop-off events of $\mathbf{r_i}$ into $\mathcal{S}_{t_j}$, subject to the taxi's capacity and delivery deadlines. The taxi, which introduces the minimum cost (*e.g.*, the minimum increase on travel cost), is normally chosen to serve request $\mathbf{r_i}$.

The searching radius γ can be gradually increased [42] and above two stages are repeated until one suitable taxi is selected [22]. However, we observe at least two limitations of existing works, which will affect their efficiency and practicality.

- **Inefficient passenger-taxi matching.** Most of the existing schemes search candidate taxis with only the origin location of a ride request [34], [41], [42]. Even though both origin and destination are utilized for a dual-side search [21], [22], they are separately considered in these works. Besides, previous works will immediately return once a valid taxi is found [21], [22], rather than searching for the best one. Such a partial trip information based passenger-taxi matching can not filter out the invalid taxis at the beginning, while it meanwhile may also miss the best taxi that introduces the minimum cost.
- **Omitting the offline ride requests.** Existing solutions mainly consider online ride requests [13], [21], [22], [41], [42], however, in practice there are still a lot of people who will hail a taxi at roadside with no submitted requests [16], [38]. According to a recent taxi service research report [3], the statistic on users' preferences of getting taxi services shows that 44.61% users only prefer online booking, while 13.71% users hail a taxi at roadside merely. In addition, 41.68% users prefer either online

TABLE I
SUMMARY OF KEY NOTATIONS.

Notation	Description
$\mathbf{G(V,E)}$	The directed graph of a road network
$cost(\cdot)$	A function to calculate the travel cost
$\mathbf{r_i}$	The i-th ride request
$\bar{\mathbf{r_i}}$	The i-th offline ride request
o_{r_i}	The origin of ride request $\mathbf{r_i}$
d_{r_i}	The destination of ride request $\mathbf{r_i}$
e_{r_i}	Delivery deadline of ride request $\mathbf{r_i}$
$\mathbf{t_j}$	Status of the j-th taxi
$\mathcal{S}_{t_j}$	Schedule of taxi $\mathbf{t_j}$
$\mathcal{R}_{t_j}$	Route of taxi $\mathbf{t_j}$
$\mathbb{P}$	A set of map partitions $\{P_z\}_{z=1}^{\kappa}$
ℓ_z	Landmark of partition P_z
$\mathbf{G}_\ell(\mathbf{V}_\ell, \mathbf{E}_\ell)$	The landmark graph
$\vec{\mathbf{v}}$	A mobility vector
C_a	A mobility cluster
γ	Searching range
$\mathbb{T}_{r_i}$	Candidate taxi set for ride request $\mathbf{r_i}$

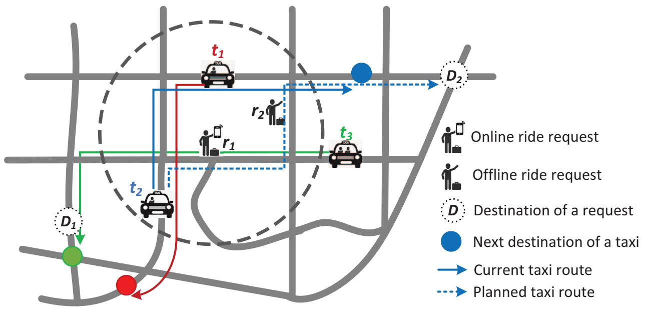

Fig. 1. An motivation example with 3 shared taxis, 1 online ride request, and 1 offline ride request. Solid lines are the current taxi routes, and the dashed line is a planned route. Taxi routes are differentiated by colors.

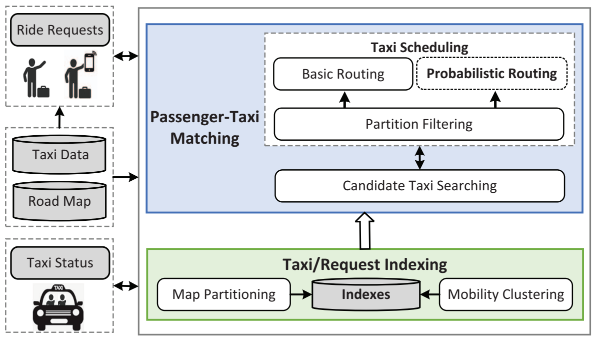

Fig. 2. The system architecture of *mT-Share*.

booking or offline hailing. From these statistics, we see that the amount of potential offline ride requests is quite large in practice, and thus such offline requests should be considered and well served by taxi ridesharing systems.

Figure 1 further explains above arguments. In this example, we assume 3 shared taxis (*i.e.*, t_1, t_2, and t_3) travel along their routes, and meanwhile passenger r_1 submits her ride request to the server while passenger r_2 prefers to hail a taxi at roadside. When receiving the request from r_1, existing schemes determine candidate taxis with a searching radius around the origin of r_1, where t_1 and t_2 are returned in this example. Then their schedules are investigated, which involves extensive computations. In fact, we find that t_2 should never be taken into consideration as it travels inversely with r_1, which introduces unnecessary computations. Although t_1 could detour to pick up r_1, we find that t_3 turns to be the best taxi to serve r_1, with no detour cost at all. However, t_3 is even not included into the initial searching results for r_1. On the other hand, since r_2 does not explicitly report her request, no taxi will serve r_2 according to all existing schemes. If the existence of ride request r_2 is perceived, we can serve r_2 as well by slightly adjusting t_2's route.

C. Problem Definition

In this paper, we consider the *mobility-aware taxi ridesharing* (MTR) problem, which is formally defined as follows.

Definition 6: (**Mobility-aware Taxi Ridesharing problem, MTR**) *Given a set of ride requests, including online requests and offline requests to predict, and a set of taxis on a road network* **G**, *the MTR problem aims to arrange ride requests to proper shared taxis, such that the number of served ride requests is maximized while the total detour cost is minimized. The arrangements should also meet the following constraints:*

- *Capacity constraint: The number of passengers in a taxi can not exceed the taxi's capacity at any time;*
- *Time constraint: The passengers of any ride request should be served within the specific deadline.*

Challenges. Different from existing works, our MTR problem not only considers online ride requests, but also takes offline ride requests into consideration to serve more passengers. Ridesharing has been proved to be NP-hard [5], [8], [34], thus the MTR problem that involves both online and offline ride requests is even more challenging. To solve this problem, we have to address at least the following two challenges.

First, both taxis and ride requests are dynamically updated in a city, which requires that the taxi schedule and route should be efficiently and wisely planned so as to guarantee the quality of service, *e.g.*, minimizing the detour cost.

Second, it is difficult to match an offline ride request with a proper shared taxi, as the exact information of offline ride requests are unknown beforehand. Such an uncertainty makes the taxi scheduling and routing to be further complicated.

IV. System Design

In this section, we will present the overview of *mT-Share*, and then detail each component in the following subsections.

A. Overview

The system architecture of *mT-Share* is illustrated in Figure 2. At a high level, *mT-Share* takes the input from the road map, historical taxi data, and real-time ride requests and taxi statuses, and dynamically arranges shared taxis to serve both online and offline ride requests. On the passenger side, a user can either explicitly submit a ride request to *mT-Share* or hail a shared taxi at roadside in an offline manner. On the taxi side, a taxi will continuously upload its status (including location, available seats, *etc.*) to the server and will receive the updated schedule and route from the server.

mT-Share consists of two major modules, *i.e.*, *Taxi/Request Indexing* and *Passenger-Taxi Matching*. Specifically, the *Taxi/Request Indexing* module makes use of mobility patterns mined from massive taxi data to divide a road map into partitions, and meanwhile groups ride requests and taxis by clustering on their travel directions. Both partitions and clusters are used to index and track the taxis. Built on such indexes, the *Passenger-Taxi Matching* module searches candidate taxis and determines the best taxi to serve a ride request. Both basic routing and probabilistic routing are supported by *mT-Share*, and they are accelerated by the partition filtering. The probabilistic routing is specially devised to allow a taxi to meet *suitable* offline ride requests with high probability.

Different from existing works [21], [22], [29], [30], [34] that index taxis/requests only using spatial grids of the road network, *mT-Share* makes use of both geographical locations and travel directions of taxis and requests for better indexing. With respect to the passenger-taxi matching, *mT-Share* differs from previous works by additionally considering the offline ride requests with a novel probabilistic routing.

B. Taxi/Request Indexing

mT-Share indexes and tracks ride requests and taxis from two aspects of both geographical location and travel direction, which are realized by conducting the bipartite map partitioning and mobility clustering, respectively.

1) Bipartite map partitioning: Rather than segmenting the road network with only geographical information [21], [22] or popular pick-up locations [41], *mT-Share* groups road network vertices into clusters according to their geographical locations and transition patterns hidden in the historical taxi data. The procedure of our map partitioning is as follows:

① *Geo-clustering.* This step classifies the vertices of a road network graph into κ spatial clusters according to their geographical locations (*i.e.*, latitude and longitude) by using the *K-mean* clustering algorithm. The vertices in a spatial cluster are geographically close. In the first time, this step is applied to all vertices. For the later times, geo-clustering is conducted on each transition cluster, which is derived in step ③, proportionally. Specifically, given a transition cluster of size n, its vertices are grouped into $\lfloor \frac{n\kappa}{N} + \frac{1}{2} \rfloor$ spatial clusters, where N is the total number of vertices in $\mathbf{V}$.

② *Transition probability calculation.* Based on the κ spatial clusters obtained in step ①, we calculate a transition probability vector $\vec{B_i}$ of size κ for each vertex v_i. Each item B_{ij} ($i = 1, 2, \cdots, N$ and $j = 1, 2, \cdots, \kappa$) is the transition probability of passengers who had taken taxis at vertex v_i and traveled to any vertex within the j-th spatial cluster. The transition probabilities are calculated with historical taxi data.

③ *Transition clustering.* We view vector $\vec{B_i}$ as a mobility feature of vertex v_i, and group all vertices into k_t transition clusters according to their transition probability vectors using the *K-mean* clustering algorithm, where $k_t < \kappa$. The vertices in a transition cluster share the similar transition patterns. We empirically set $k_t = 20$ for *mT-Share*.

We repeat above three steps until the κ spatial clusters derived in step ① do not change. We treat these clusters as the final *partitions* of graph $\mathbf{G}$, denoted as $\mathbb{P} = \{P_z\}_{z=1}^{\kappa}$. The vertices in each partition are both spatially close and highly similar in transition patterns. Such properties facilitate the predictions of offline ride requests. Figure 3(b) shows the result when we apply bipartite map partitioning on the road network graph of Chengdu city shown in Figure 3(a).

For each partition, we compute a landmark, which can be viewed as the center of a partition. Based on all partitions and their landmarks, we build a landmark graph $\mathbf{G}_\ell$, which can be used to accelerate the route planning later.

***Definition** 7:* (**Landmark**) *The landmark of a partition P_z is the vertex $\ell_z \in P_z$ that has the minimum total distance to all other vertices of partition P_z.*

***Definition** 8:* (**Landmark Graph**) *A landmark graph is denoted by $\mathbf{G}_\ell(\mathbf{V}_\ell, \mathbf{E}_\ell)$, where vertices in $\mathbf{V}_\ell$ are landmarks of all partitions, and each edge in $\mathbf{E}_\ell$ is formed between any two landmarks if their corresponding partitions are adjacent.*

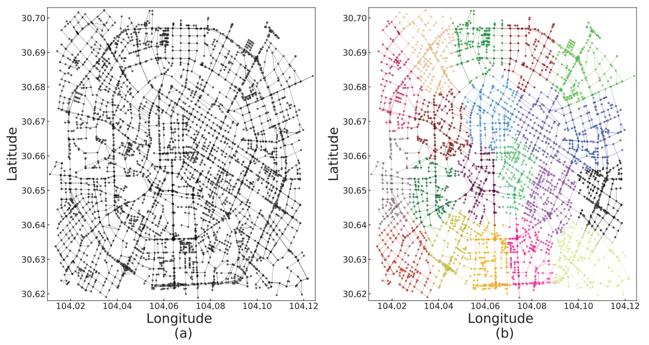

Fig. 3. (a) The road network of Chengdu city, China. (b) The result of bipartite map partitioning applied on (a), where we set $\kappa = 20$ just for a clear demonstration and the spatial partitions are differentiated by colors.

2) Mobility clustering: Different from previous works that only use locations to index ride requests and taxis [21], [22], [29], [30], [34], [41], [42], *mT-Share* models both ride requests and taxis as *mobility vectors* and further groups them through mobility clustering on their travel directions.

***Definition** 9:* (**Mobility Vector**) *A mobility vector $\vec{\mathbf{v}}$ is defined as a vector pointing from an origin (lat_o, lng_o) to a destination (lat_d, lng_d), denoted by $\vec{\mathbf{v}} = (lat_o, lng_o, lat_d, lng_d)$.*

For a ride request $\mathbf{r_i}$, we create its mobility vector $\vec{\mathbf{v}}_{r_i}$ with its origin o_{r_i} and destination d_{r_i}. For a taxi $\mathbf{t_j}$ with m passengers $\{\mathbf{r_i}\}_{i=1}^{m}$, we regard its current location loc_{t_j} as the origin location of mobility vector $\vec{\mathbf{v}}_{t_j}$, and take the center of all destinations of its passengers (*i.e.*, $\frac{\sum_{i=1}^{m} d_{r_i}}{m}$) as the destination of $\vec{\mathbf{v}}_{t_j}$. Empty taxi are not considered for mobility clustering.

Based on the mobility vectors of all ride requests and taxis, we group them into clusters. The first ride request will form the initial cluster, and the subsequent ride requests will join existing clusters or form new clusters individually. For each cluster C_a, we maintain a *general mobility vector* $\vec{\mathbf{v}}_{C_a}$, whose origin and destination are averagely calculated from the origins and destinations of all cluster members. When a new ride request $\mathbf{r_i}$ arrives, its mobility vector is compared to each general mobility vector. $\mathbf{r_i}$ is included into cluster C_a if their travel direction difference is sufficiently small. Specifically, we use the *cosine similarity* as the distance metric to measure their travel direction difference θ, *i.e.*,

$$\cos(\theta) = \frac{\vec{\mathbf{v}}_{r_i} \cdot \vec{\mathbf{v}}_{C_a}}{||\vec{\mathbf{v}}_{r_i}|| \times ||\vec{\mathbf{v}}_{C_a}||}. \tag{1}$$

When $\cos(\theta) \geq \lambda$ where λ is a predefined threshold, we consider $\mathbf{r_i}$ travels along a similar direction with the passengers in cluster C_a and they might share the same taxi. Otherwise, $\mathbf{r_i}$ will form a new mobility cluster.

The mobility clusters and their corresponding general mobility vectors are dynamically updated when ride requests have been completed or new ride requests are received.

3) Index of taxis: To facilitate the taxi searching for a given ride request, *mT-Share* builds index structures based on both map partitions and mobility clusters.

- *Map partition based indexing*. For each partition P_z, we maintain a nearby taxi list $P_z.L_t$ to record the IDs of taxis that are now in or will arrive at partition P_z within a time threshold (*e.g.*, 1 hour). The taxi IDs are sorted in an ascending order according to their estimated arrival times. Note that $P_z.L_t$ should be updated dynamically.
- *Mobility cluster based indexing*. For each mobility cluster C_a, we maintain a taxi list $C_a.L_t$ as well. This list contains the IDs of taxis that are now serving passengers and meanwhile traveling in a similar direction. Note that $C_a.L_t$ should be dynamically updated once C_a changes.

Memory complexity. According to above indexing structures, each taxi is indexed by multiple map partitions and (at most) one mobility cluster. Besides, each ride request is indexed by one mobility cluster. Therefore, the total memory complexity of indexing is $\mathcal{O}((x+1)M+R)$, where M is the number of taxis, x is the number of partitions a taxi can visit within the time threshold, and R is the number of all requests.

C. Passenger-Taxi Matching

For each new ride request $\mathbf{r_i}$, *mT-Share* firstly determines a set of candidate taxis by exploiting above indexes, and then heuristically checks their schedules to select the most suitable one, which should introduce the minimum detour cost.

1) Candidate taxi searching: Different from previous works that gradually increase the searching range and immediately return a valid (but may not be the best) taxi [21], [22], [41], [42], *mT-Share* determines the best taxi to serve each request $\mathbf{r_i}$ in an aggressive manner. Specifically, we set the taxi searching range γ of $\mathbf{r_i}$ as the product of a typical driving speed and a waiting time Δt, which is calculated as

$$\Delta t = e_{r_i} - cost(o_{r_i}, d_{r_i}) - t_{r_i}, \tag{2}$$

where $e_{r_i} - cost(o_{r_i}, d_{r_i})$ is the pick-up deadline and t_{r_i} is the release time of $\mathbf{r_i}$. Centering at o_{r_i}, we get a set of partitions $\mathbb{S}_{r_i}$ intersected with the searching area. For each partition $P_z \in \mathbb{S}_{r_i}$, we retrieve its taxi list $P_z.L_t$. Besides, by comparing $\mathbf{r_i}$'s mobility vector to all existing mobility clusters, we may find a cluster C_a sharing the similar travel direction with $\mathbf{r_i}$. Therefore, candidate taxi set $\mathbb{T}_{r_i}$ for request $\mathbf{r_i}$ is derived as

$$\mathbb{T}_{r_i} = \{\cup_{P_z \in \mathbb{S}_{r_i}} P_z.L_t\} \cap C_a.L_t \tag{3}$$

The set $\mathbb{T}_{r_i}$ can be further refined by: (i) including the empty taxis $\{\mathbf{t_j}\}$, where $\mathbf{t_j} \in P_z.L_t$ ($P_z \in \mathbb{S}_{r_i}$) and $\mathcal{S}_{t_j} = \emptyset$; (ii) filtering out the taxis with no available seats; (iii) filtering out the taxis that cannot arrive $\mathbf{r_i}$'s locating partition P_i before the pick-up deadline by exploiting the taxi arrival time recorded in $P_i.L_t$. These operations could filter out invalid taxis at the beginning and thus save lots of computations.

2) Taxi scheduling: Given the candidate taxi set $\mathbb{T}_{r_i}$, taxi scheduling aims to select the most suitable taxi that can serve ride request $\mathbf{r_i}$ while introducing the minimum detour cost. In principle, we should rearrange the events of a taxi schedule $\mathcal{S}_{t_j}$ after incorporating the pick-up event o_{r_i} and drop-off event d_{r_i} of $\mathbf{r_i}$. However, it will introduce extensive computations and is prohibited in practice. Instead, *mT-Share* will insert o_{r_i} and d_{r_i} into $\mathcal{S}_{t_j}$, while keeping the existing schedule unchanged. This is a typical design choice, similar as previous works [21], [22], [34], [41]. The feasibility of inserting $\mathbf{r_i}$ into a schedule is mainly determined by the time constraints of ride requests, which is finally affected by the route planning.

Algorithm 1: Taxi Scheduling

```
1  Input: Ride request $\mathbf{r_i}$ and candidate taxi set $\mathbb{T}_{r_i}$;
2  Output: A taxi with updated schedule/route for $\mathbf{r_i}$;
3  foreach taxi $\mathbf{t_j} \in \mathbb{T}_{r_i}$ do
4      foreach schedule instance $\mathcal{S}'_{t_j} \leftarrow \{\mathcal{S}_{t_j}, o_{r_i}, d_{r_i}\}$ do
5          if $flag$ then
6              $\mathcal{R}'_{t_j}$ = ProbabilisticRouting($\mathcal{S}'_{t_j}$, $\mathbf{t_j}$);
7          else
8              $\mathcal{R}'_{t_j}$ = BasicRouting($\mathcal{S}'_{t_j}$, $\mathbf{t_j}$);
9          $\omega = cost(\mathcal{R}'_{t_j}) - cost(\mathcal{R}_{t_j})$;
10 Select the taxi schedule instance with the minimum $\omega$;
```

We present the taxi scheduling algorithm in **Algorithm 1**. For each candidate taxi $\mathbf{t_j} \in \mathbb{T}_{r_i}$, we enumerate all possible schedules by inserting event o_{r_i} and event d_{r_i} into $\mathcal{S}_{t_j}$, where o_{r_i} should be ahead of d_{r_i}. For each schedule instance $\mathcal{S}'_{t_j}$ of $\mathbf{t_j}$, we plan a route and calculate the detour cost as

$$detour\, cost = cost(\mathcal{R}'_{t_j}) - cost(\mathcal{R}_{t_j}), \tag{4}$$

where $\mathcal{R}'_{t_j}$ is the updated route of $\mathcal{R}_{t_j}$ after picking up $\mathbf{r_i}$. We run this operation for all schedule instances of all candidate taxis, and select the taxi with the minimum detour cost as the best one to serve $\mathbf{r_i}$ following the updated schedule and route.

mT-Share serves both online and offline ride requests, which is achieved by the routing algorithms. If a taxi has sufficient empty seats while there are few online ride requests right now, the taxi driver can enable probabilistic routing that will plan a route to meet the suitable offline ride requests with high probability. Specifically, if the indicator $flag$ in **Algorithm 1** is true, *mT-Share* invokes function $ProbabilisticRouting()$ to plan a probabilistic route for seeking offline ride requests. Otherwise, $BasicRouting()$ is invoked, which returns the shortest path. Both functions are optimized with $PartitionFilter()$ that prunes the searching space for the route planning.

Since route planning is usually the efficiency bottleneck of taxi scheduling [21], [34], *mT-Share* thus proposes a two-phase route planning to optimize both basic routing and probabilistic routing. Given a schedule instance $\mathcal{S}'_{t_j}$, *mT-Share* plans the route for each consecutive event pair $(s_z, s_{z+1}) \in \mathcal{S}'_{t_j}$ through two phases, *i.e.*, partition filtering that reduces the searching space for route planning and segment-level routing that returns the final travel path. By concatenating the travel paths between consecutive events (*i.e.*, the operation $\bowtie$ in **Algorithm 3** and **Algorithm 4**), we can derive the final route $\mathcal{R}'_{t_j}$ for schedule $\mathcal{S}'_{t_j}$. The two phases are detailed as follows.

– Phase 1: Partition filtering

Algorithm 2: Partition Filtering

```
1 Function PartitionFilter($s_z, s_{z+1}$):
2     Find partition $P_z$, $P_{z+1}$, and landmark $\ell_z$, $\ell_{z+1}$;
3     $\mathcal{P} \leftarrow \emptyset$;
4     foreach $P_i \in \mathbb{P}$ do
5         if $P_i$ satisfies the two rules then
6             $\mathcal{P} = \mathcal{P} \cup \{P_i\}$;
7     return $\mathcal{P}$;
```

Algorithm 3: Basic Routing

```
1 Function BasicRouting($\mathcal{S}$, $\mathbf{t_j}$):
2     $\mathcal{R} \leftarrow \emptyset$;
3     for $z = 1$ to $(|\mathcal{S}| - 1)$ do
4         $\mathcal{P}$ = PartitionFilter $(s_z, s_{z+1})$;
5         Build subgraph $\mathbf{G}_z$ from $\mathcal{P}$;
6         Find the shortest path $R_z$ using the Dijkstra's
            algorithm on $\mathbf{G}_z$;
7         $\mathcal{R} = \mathcal{R} \bowtie R_z$;
8     return $\mathcal{R}$;
```

This phase is executed on the landmark graph $\mathbf{G}_\ell$. For any two consecutive events $(s_z, s_{z+1}) \in \mathcal{S}'_{t_j}$, we derive their locating partitions P_z and P_{z+1}, and corresponding landmarks ℓ_z and ℓ_{z+1}. We use the travel cost between ℓ_z and ℓ_{z+1}, *i.e.*, $cost(\ell_z, \ell_{z+1})$, to approximate the length of the shortest path between s_z and s_{z+1}. In addition, we use the two landmarks to generate a mobility vector $\vec{\mathbf{v}}_z$. Then we examine each partition $P_i \in \mathbb{P}$ with the following two rules:

- *Travel direction rule.* The direction difference θ between the mobility vector pointing from ℓ_z to ℓ_i and $\vec{\mathbf{v}}_z$ is sufficiently small, *i.e.*, $\cos(\theta) \geq \lambda$.
- *Travel cost rule.* Travel cost of the path that links ℓ_z and ℓ_{z+1} via ℓ_i is not remarkably greater than the travel cost of the shortest path between ℓ_z and ℓ_{z+1}, *i.e.*,

$$cost(\ell_z, \ell_i) + cost(\ell_i, \ell_{z+1}) \leq (1 + \varepsilon) \times cost(\ell_z, \ell_{z+1}),$$

  where ε is a predefined parameter and is conservatively set as 1.0 in this paper.

Partitions that meet above rules are retained into a set $\mathcal{P}^z_{t_j}$. The pseudocode of partition filtering is listed in **Algorithm 2**. Figure 4(a) demonstrates the partition filtering for event pair (s_z, s_{z+1}), where the gray partitions are retained in $\mathcal{P}^z_{t_j}$.

– Phase 2: Segment-level routing

Rather than planning the route for a schedule instance $\mathcal{S}'_{t_j}$ on the original graph $\mathbf{G}$, *mT-Share* will find the feasible route on a much smaller subgraph, which is constructed by the vertices and edges belonging to the partitions derived from partition filtering. The reduced searching space can significantly improve the computation efficiency of route planning, and thus both basic routing and probabilistic routing are executed on the reduced graph. Next we introduce the two routing modes.

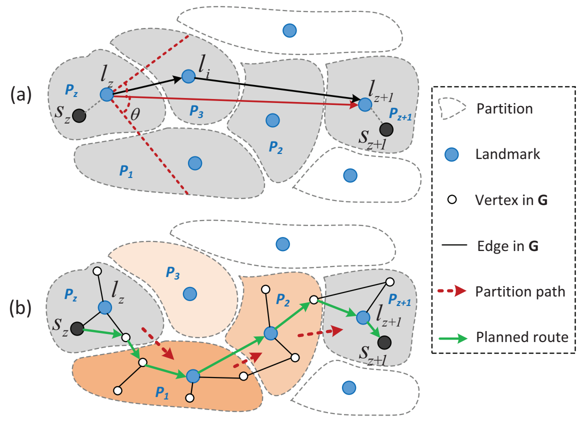

Fig. 4. (a) Illustration of partition filtering with the constraints on travel direction and travel cost. The gray partitions are retained while the white ones are filtered out. (b) Probabilistic routing on $\mathcal{P}'_{t_j}$, where red color of partitions indicates the probability (the darker the higher). A valid route is planned based on partition path $\mathcal{H}'_{t_j} = \{P_z, P_1, P_2, P_{z+1}\}$.

• *Basic routing.* It aims to find the shortest path for any two consecutive events in a schedule instance. The shortest path based route planning has been frequently used by previous works for ridesharing routing [21], [22], [34], [41]. In *mT-Share*, the *Dijkstra*'s algorithm [10] is used to calculate the shortest path on the subgraph. The pseudocode of basic routing is listed in **Algorithm 3**.

• *Probabilistic routing.* It supports a taxi to opportunistically encounter suitable offline ride requests. Here a ride request $\mathbf{r_i}$ is considered as *suitable* if $\mathbf{r_i}$ travels with similar direction as the given taxi. In theory, we should calculate the probabilities of suitable requests over all graph vertices and plan a route that accumulates the maximum probability of picking up new suitable passengers. However, it turns to be computationally prohibitive, which has been proved to be NP-Complete [16].

To avoid the huge computation overheads, *mT-Share* plans the probabilistic route in an heuristic manner. For any two consecutive events (s_z, s_{z+1}) in a schedule instance, *mT-Share* further refines the partition set $\mathcal{P}^z_{t_j}$ with transition patterns and plans the probabilistic route on a smaller subgraph, which is built from the refined partition set. The algorithm is presented in **Algorithm 4**, and the main steps are as follows:

① *Probability calculation of suitable passengers.* For each partition $P_i \in \mathcal{P}^z_{t_j}$ and the given candidate taxi $\mathbf{t_j}$, we select the destination partitions of suitable passengers. Specifically, for each partition $P_a \in \mathbb{P}$, we keep P_a into set $\mathcal{P}_d$ only if the travel direction difference θ between taxi $\mathbf{t_j}$ and the mobility vector constructed by the landmarks of P_i and P_a is sufficiently small, *i.e.*, $\cos(\theta) \geq \lambda$. After obtaining set $\mathcal{P}_d$, we calculate the probability π_i of meeting suitable passengers within partition P_i by accumulating the transition probabilities of each vertex $v_c \in P_i$ to each potential destination in set $\mathcal{P}_d$. The transition probabilities of each vertex to all partitions have been calculated during bipartite map partitioning (see Section IV-B1) and they can be cached for reuse to save computations.

② *Partition path planning.* A landmark subgraph $\mathbf{G}^z_\ell$ is built

Algorithm 4: Probabilistic Routing

```
1  Function ProbabilisticRouting($\mathcal{S}$, $\mathbf{t_j}$):
2      $\mathcal{R} \leftarrow \emptyset$;
3      for $z = 1$ to $(|\mathcal{S}| - 1)$ do
4          $\mathcal{P}$ = PartitionFilter $(s_z, s_{z+1})$;
5          Calculate probability $\pi_i$ of meeting suitable
             offline requests for partition $P_i \in \mathcal{P}$;  ▷ step ①
6          Build weighted landmark subgraph $\mathbf{G}_\ell^z$ from $\mathcal{P}$;
7          $attempt = 0$;
8          while true do
9              Select the maximum weighted path from $\ell_z$
                 to $\ell_{z+1}$ on $\mathbf{G}_\ell^z$ to form the partition path
                 $\mathcal{H}$;                              ▷ step ②
10             Build weighted subgraph $\mathbf{G}_z$ from $\mathcal{H}$;
11             Find the shortest path $R_z$ using the
                 Dijkstra's algorithm on $\mathbf{G}_z$;   ▷ step ③
12             $attempt = attempt + 1$;
13             if $R_z$ is valid then
14                 break;
15             else if $attempt > 5$ then
16                 return $\emptyset$;
17         $\mathcal{R} = \mathcal{R} \bowtie R_z$;
18     return $\mathcal{R}$;
```

from the landmarks and edges of all partitions in $\mathcal{P}_{t_j}^z$, where each landmark vertex is annotated with the probability π_i as the weight. Since the weighted graph is usually small, we will enumerate all paths that link source and destination partitions to find the maximum weighted path. By mapping landmark vertices of the selected path to their corresponding partitions, we obtain a partition path $\mathcal{H}_{t_j}^z$, which travels from P_z to P_{z+1} and accumulates the maximum probability at partition-level.

③ *Fine-grained route planning over partition path.* We build a weighted subgraph $\mathbf{G}_z$ with vertices and edges from partitions of $\mathcal{H}_{t_j}^z$, where each vertex v_c is annotated with a weight $\frac{1}{\psi_c}$ ($\psi_c > 0$). We set ψ_c as the total transition probability from vertex v_c to the destination partition set $\mathcal{P}_d$ of v_c's locating partition. We search a path using the *Dijkstra*'s algorithm [10] on graph $\mathbf{G}_z$, which has the minimum weights while satisfying the constraints on deadlines of passengers already in taxi $\mathbf{t_j}$. We regard this path as the final taxi route R_z, which has the largest probability to encounter suitable offline passengers between (s_z, s_{z+1}).

If we do not find a valid route in step ③, the sub-optimal partition path on $\mathcal{P}_{t_j}^z$ will be returned in step ②. The latter two steps are repeated until we derive a valid route for event s_z and s_{z+1} (*i.e.*, meeting the passengers' deadlines) or the predefined number of attempts (*e.g.*, 5) is achieved. Otherwise, the schedule instance is marked as invalid since there exists no available route to linking the two events. Similarly, we concatenate the paths of any two consecutive events to form the final route. Figure 4(b) demonstrates the partition path and final path for two consecutive events.

Since the probabilistic routing is more time consuming than the basic routing, a taxi driver may enable it only when there is sufficient idle capacity and online ride requests are inadequate. When a taxi encounters an offline ride request $\bar{\mathbf{r_i}}$, it can serve this request only when a valid schedule exists. Otherwise, the taxi driver will report this offline request to the server and *mT-Share* will assign another proper taxi to serve $\bar{\mathbf{r_i}}$. Although the interaction may introduce slight delays, it potentially brings benefits for both taxi drivers and offline passengers. Therefore, it is acceptable for taxi drivers to take a chance for hunting offline ride requests at the idle time.

Time complexity. To process a request $\mathbf{r_i}$, *mT-Share* needs to examine all schedule instances of all candidate taxis in $\mathbb{T}_{r_i}$ and then selects the one with the minimum detour cost, which involves above four algorithms. If basic routing is used, the time complexity is $\mathcal{O}(|\mathbb{T}_{r_i}|m^3\kappa)$, where $|\mathbb{T}_{r_i}|$ is the size of candidate taxi set, m is the number of events in a schedule instance, and κ is the number of map partitions in $\mathbb{P}$. If probabilistic routing is adopted, the time complexity becomes $\mathcal{O}(\frac{|\mathbb{T}_{r_i}|m^3ND|\mathcal{P}|}{\kappa})$, where D is the size of historical taxi data for request probability calculations, $|\mathcal{P}|$ is the number of partitions returned by **Algorithm 2**, and $\frac{N}{\kappa}$ represents the average number of vertices in a partition. The above analysis assumes a shortest path query takes $\mathcal{O}(1)$ time as with many previous studies [8], [34], since the shortest paths between any two vertices can be pre-calculated and cached.

V. Performance Evaluation

In this section, we will conduct experiments to evaluate *mT-Share* with a large real-world taxi dataset.

A. Experimental Setup

Dataset. We conduct experimental evaluations using a real-world taxi dataset, which is publicly released by Didi's GAIA initiative [1]. This dataset contains 7065907 taxi transactions that are collected from Chengdu city, China, in 2016, which has been used by previous works [15], [34]. Each transaction consists of a transaction ID, a taxi ID, and a ride request, which includes a pick-up latitude/longitude, a drop-off latitude/longitude, and a release time. We use data of two specific time periods to simulate two typical ridesharing scenarios:

- *Peak scenario.* During the peak hours, taxis usually have to process a large volume of online ride requests, and thus the offline ride requests are omitted. We use the data from 8:00AM-9:00AM of a workday, which contains the most hourly ride requests, *i.e.*, 29534, to test the performance of *mT-Share* in the peak hours.
- *Non-peak scenario.* During the non-peak hours, there are usually inadequate online ride requests while most taxis have sufficient idle capacities. Therefore, taxi drivers can utilize probabilistic routing to opportunistically seek offline requests. In the experiments, we assume a taxi that has more than half of the capacity in idle can enable the probabilistic routing mode. We use the taxi data

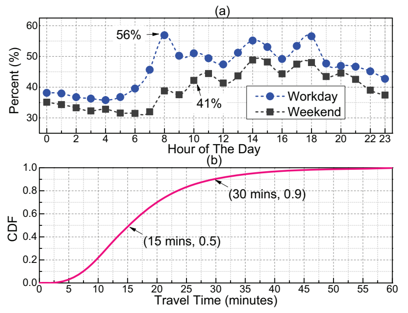

Fig. 5. Statistics of the taxi dataset.

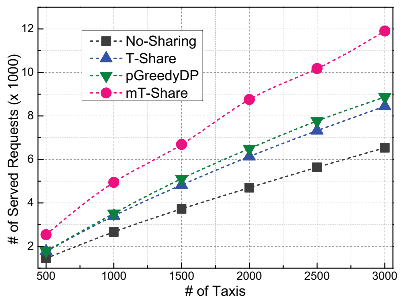

Fig. 6. Results on served requests in *peak scenario*.

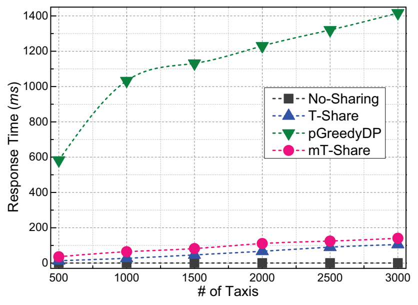

Fig. 7. The response time in *peak scenario*.

from 10:00AM-11:00AM of a weekend, which consists of 15480 ride requests, to evaluate *mT-Share* in the non-peak hours. To simulate the offline ride requests, we randomly select 5000 requests and make their origin, destination, and release time to be invisible to the ridesharing system.

The remaining taxi data in the dataset are used for bipartite map partitioning and ride request probability calculation. Figure 5 plots some statistics about the dataset. Figure 5(a) shows the average taxi utilization in workdays and weekends, where the utilization of a taxi is calculated as the proportion of the time serving passengers within each hour. Specifically, the utilization rates of 8:00AM-9:00AM in workdays and 10:00AM-11:00AM in weekends are 56% and 41%, respectively. Figure 5(b) further presents the travel time distribution of all taxi transactions, where the 50-percentile and 90-percentile travel times are 15 minutes and 30 minutes, respectively.

We use the open-sourced OpenStreetMap [2] to export roads of Chengdu city, China, and present the road network as a directed graph $\mathbf{G}(\mathbf{V}, \mathbf{E})$ that consists of 214440 vertices and 466330 edges, covering an area of more than $70\,km^2$. Figure 3(a) shows the experimental road network.

Compared schemes. We will compare *mT-Share* with the following schemes.

(1) *No-Sharing* works with no ridesharing and assigns a ride request to the nearest vacant taxi within a searching range γ.

(2) *T-Share* [21], [22], one of the state-of-the-art methods, indexes all taxis and ride requests with grids and determines the candidate taxis using a dual-side search with an adaptive searching range γ from both origin and destination of a ride request. It will immediately return if a valid candidate is found.

(3) *pGreedyDP* [34], one of the state-of-the-art methods, indexes all taxis and ride requests with grids like *T-Share*, and selects the candidate taxis within a searching range γ around the origin of ride request $\mathbf{r_i}$. It determines the insertion of $\mathbf{r_i}$ into an existing schedule via dynamic programming to improve the efficiency of taxi scheduling.

(4) *mT-Sharepro* is the version of our scheme that enables the probabilistic routing. Since it may introduce large computation overheads, we only evaluate *mT-Sharepro* in the *non-peak scenario*, where seeking and serving offline ride requests are necessary for taxi drivers during these periods [6], [38].

We slightly adjust *T-Share* and *pGreedyDP* to serve offline passengers as well. Along the route planned by *T-Share* or

TABLE II
THE PARAMETER SETTINGS.

Parameter	Setting
Number of taxis	500, 1000, 1500, **2000**, 2500, 3000
Number of partitions κ	50, 100, **150**, 200, 250
Capacity of a vehicle	2, **3**, 4, 5, 6
Flexible factor ρ	1.1, 1.2, **1.3**, 1.4, 1.5, 1.6, 1.7, 1.8, 1.9, 2.0
Threshold λ	**0.5** (*i.e.*, $\theta = 45°$)
Taxi searching range γ	$\mathbf{2.5\,km}$

pGreedyDP, if a taxi $\mathbf{t_j}$ with sufficient idle capacity happens to encounter some suitable offline passenger $\bar{\mathbf{r}}_\mathbf{i}$, whose ride request can be validly inserted into $\mathbf{t_j}$'s schedule, then $\mathbf{t_j}$ can serve $\bar{\mathbf{r}}_\mathbf{i}$. Similarly, they will serve offline ride requests only in the *non-peak scenario*.

Performance metrics. All schemes are evaluated in terms of *number of served requests*, *response time*, and *detour time*. Response time is the time to process a ride request, while detour time is the increased travel time compared to the travel time of no ridesharing for a ride request. Both number of served requests and response time have been widely adopted in previous real-time ridesharing systems [14], [21], [22], [34].

Implementation. We implement *mT-Share* and the alternative schemes in Python. For each ride request $\mathbf{r_i}$, its origin/destination is mapped to the closest vertex in graph $\mathbf{G}$. We simulate the delivery deadline e_{r_i} with a *flexible factor* ρ, which indicates the acceptable travel cost by passengers compared to the shortest path [8]. Given the release time t_{r_i} and travel cost $cost(o_{r_i}, d_{r_i})$ of $\mathbf{r_i}$, delivery deadline e_{r_i} is

$$e_{r_i} = t_{r_i} + cost(o_{r_i}, d_{r_i}) \times \rho. \quad (5)$$

The initial location of each taxi is randomly chosen from the vertices of graph $\mathbf{G}$. When a taxi $\mathbf{t_j}$ is serving passengers, it travels following the planned schedule $\mathcal{S}_{t_j}$ and route $\mathcal{R}_{t_j}$. For simplicity, we assume the constant taxi speed as with previous works [7], [16], [44], and set it as $15\,km/h$. In addition, we fix the searching range $\gamma = 2.5\,km$, which is equivalent to the waiting time $\Delta t = 10$ minutes.

All experiments are conducted on a server with Intel Core i7-6700 CPU@3.40GHz and 16GB memory. Each experimental setting is repeated 10 times and the average results are reported. To accelerate the route planning, the shortest paths between any two vertices in $\mathbf{G}$ and the travel cost between any two landmarks in $\mathbf{G}_\ell$ are pre-calculated and cached in the memory [14], [44], which can be accessed by all schemes.

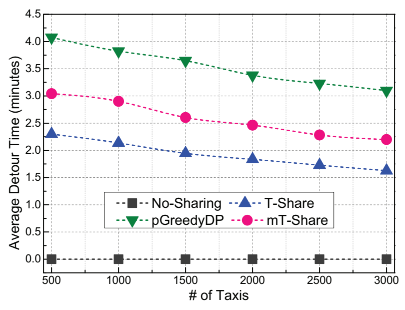

Fig. 8. The detour time in *peak scenario*.

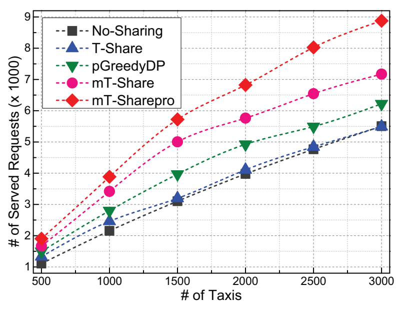

Fig. 9. The served requests in *non-peak scenario*.

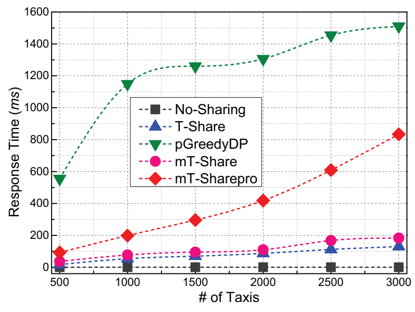

Fig. 10. The response time in *non-peak scenario*.

Table II summarizes the major parameter settings, where the default values are marked in bold.

B. Performance Comparison

We compare *mT-Share* with the alternative schemes in both *peak scenario* and *non-peak scenario* by varying the number of available taxis from 500 to 3000, with a step of 500.

Comparisons in *peak scenario*. With more available taxis, all schemes can serve more ride requests, as shown in Figure 6. Compared to *No-Sharing*, ridesharing can greatly increase the number of served requests. From Figure 6, we see *pGreedyDP* performs better than *T-Share* since it optimizes the ridesharing routing and thus can return a better request-taxi match. Among all schemes, our scheme serves the most ride requests because *mT-Share* has optimized both candidate taxi searching and ridesharing routing by fully exploiting the mobility information. For example, with 3000 taxis, the numbers of served requests of *No-Sharing*, *T-Share*, *pGreedyDP*, and *mT-Share* are 6534, 8441, 8868, and 11906, respectively. Compared to state-of-the-art methods, *i.e.*, *T-Share* and *pGreedyDP*, *mT-Share* serves 42% and 36% more ride requests, respectively.

Figure 7 shows that the response time increases with more available taxis. Due to its simplicity, *No-Sharing* can always process a request within 1 millisecond (ms). *mT-Share* takes a bit longer time to response a request than *T-Share*, while *pGreedyDP* is the slowest one. This is because the decision phase of *pGreedyDP* to calculate the low bound detour for each candidate taxi is time consuming. In general, *mT-Share* responses a ride request within $35 \sim 140\,ms$, and outperforms *pGreedyDP* by $4 \sim 10$ times on the response time.

We report the detour time in Figure 8. *No-Sharing* has no detour at all and the other schemes introduce detour time about $1 \sim 4$ minutes. More taxis allows a ridesharing scheme to find a more suitable taxi for a ride request, and thus the detour time is reduced. Overall *T-Share* introduces the minimum detour time while our scheme takes the second place and is quite close to *T-Share*. In contrary, *pGreedyDP* almost doubles the detour time of *T-Share*. Specifically, *mT-Share* improves *pGreedyDP* by 31% to 40% on the average detour time.

Comparisons in *non-peak scenario*. Figure 9 presents the number of served requests for all schemes in *non-peak scenario*. Compared to the results in Figure 6, we see the gap between ridesharing schemes and *No-Sharing* becomes smaller, since there are much fewer ride requests in the non-peak hours. Even in some settings, we see that *T-Share* almost serves the similar number of requests as *No-Sharing*. However, *mT-Share* and *mT-Sharepro* still perform much better than *T-Share* and *pGreedyDP*. Thanks to the probabilistic routing, *mT-Sharepro* can serve more offline passengers, where we see an increase about $13\% \sim 24\%$ than *mT-Share*. Compared to *T-Share* and *pGreedyDP*, on average *mT-Sharepro* serves 62% and 58% more ride requests, respectively.

We plot the response time in Figure 10. The results of *No-Sharing*, *T-Share*, *pGreedyDP*, and *mT-Share* are quite similar with their performances in *peak scenario* as shown in Figure 7. Due to the huge computation overhead of the probabilistic routing that involves probability calculations for graph vertices and then selects the route with the maximum probability, the response time of *mT-Sharepro* becomes much larger than *mT-Share*, *i.e.*, $2.5 \sim 4.5$ times slower. When we increase the number of taxis, the size of candidate taxi set becomes larger as well, and thus *mT-Sharepro* requires more time to response each request. However, *mT-Sharepro* can still response much faster than *pGreedyDP*, with $81\% \sim 497\%$ performance gain.

Results on detour time of *No-Sharing*, *T-Share*, *pGreedyDP*, and *mT-Share* shown in Figure 11 are similar with their results of *peak scenario* in Figure 8 as well. This is because their routing algorithms are the same in both scenarios. Among the five schemes, *mT-Sharepro* has the largest detour time. This is because probabilistic routing may return some longer routes for a taxi to opportunistically seek offline ride requests. Even so, we see that the detour cost difference between *mT-Sharepro* and *pGreedyDP* keeps within 0.5 minutes.

C. Detailed Evaluation

In this subsection, we study the impacts of some important parameters and design choices on ridesharing performance.

Impact of partition number κ. We conduct experiments in the *peak scenario* by varying partition number κ while keeping other parameters as the default values. Figure 12 shows that the number of served requests for the three schemes increases initially and then decreases beyond some value, *e.g.*, $\kappa = 150$. This is because both too small or oversize κ will result in a reduced candidate taxi set, and thus affect the ridesharing performance. Taking *mT-Share* as an example, when we vary κ from 50 to 150, the number of candidate taxis increases by

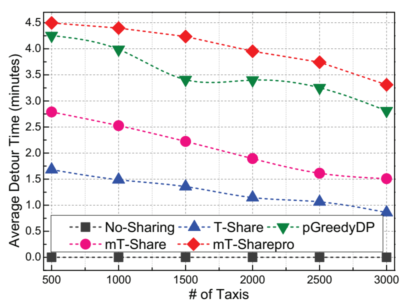

Fig. 11. The detour time in the *non-peak scenario*.

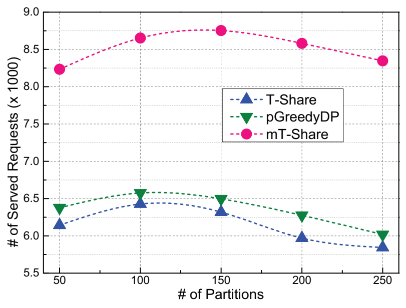

Fig. 12. Impact of the number of partitions κ.

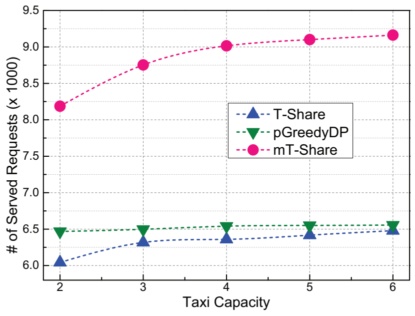

Fig. 13. Impact of the taxi capacity.

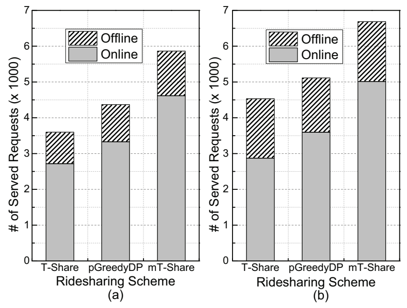

Fig. 14. Impact of the routing schemes: (a) Basic routing; (b) Probabilistic routing.

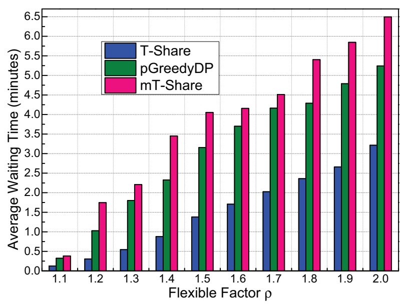

Fig. 15. Impact of ρ on the average waiting time of passengers in the *peak scenario*.

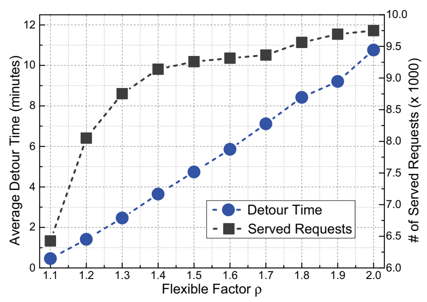

Fig. 16. Impact of ρ on average detour time and number of served requests.

17% and the number of served requests increases from 8234 to 8753. With even more partitions, we find the candidate taxi set is shrinking, and thus the total served requests are reduced.

Impact of capacity. This study is also conducted in the *peak scenario* with default parameters. When we increase the capacity of a taxi, the same number of taxis will have more supplies and thus can serve more ride requests. Figure 13 shows the number of served requests increases with larger taxi capacity. However, the impact of capacity on *pGreedyDP* is quite limited, while *mT-Share* benefits a lot. For example, compared to $capacity = 2$, *mT-Share* can serve 12% more ride requests when $capacity = 6$.

Impact of routing schemes. We also study the benefit of probabilistic routing for seeking offline ride requests. Specifically, we combine basic routing or probabilistic routing with *T-Share*, *pGreedyDP*, and *mT-Share*, and run these schemes in *non-peak scenario* with default parameters. Figure 14 shows the compositions of served requests for different combinations. Although the schemes with basic routing could meet some offline requests by chance (see Figure 14(a)), probabilistic routing can further enlarge the probability as shown in Figure 14(b). By comparing the results in Figure 14(a) and 14(b), we find that about 89%, 46%, and 34% more offline requests are successfully predicted by probabilistic routing for *T-Share*, *pGreedyDP*, and *mT-Share*, respectively. Overall, the three schemes can serve 26%, 17%, and 14% more requests, respectively, by exploiting the mobility patterns for route planning.

Impact of flexible factor ρ. We study the impact of flexible factor ρ in *peak scenario* with default parameters. We firstly study the impact of ρ on passengers' waiting time, which is calculated as the difference between the time a taxi picks up the passengers and the time passengers release the ride request. Figure 15 plots the average waiting time of the ridesharing schemes when we vary ρ. A larger ρ means that passengers are more tolerant on the delivery time, and thus longer waiting time is observed for all schemes. In general, *T-Share* has the shortest waiting time, while *mT-Share* has the relatively longer waiting time. The gap between *pGreedyDP* and *mT-Share* is quite small, *i.e.*, $0.05 \sim 1.25$ minutes. If we further consider the detour time as shown in Figure 8, we find that *mT-Share* actually has similar service quality as *pGreedyDP*.

Parameter ρ indicates the tolerant detour cost by passengers, and we find the average detour time gradually increases with the increase of ρ, as shown in Figure 16. Larger ρ allows a taxi to serve more ride requests, while the number of served requests increases slightly beyond $\rho = 1.3$. However, more detours bring about negligible benefit. For example, the number of served requests and detour time are 8753 and 2.5 minutes when $\rho = 1.3$, while they increase to 9140 and 3.6 minutes when $\rho = 1.4$. It means 4% improvement on served requests comes at the expense of 48% increases of detour cost.

VI. Conclusion

In this paper, we present *mT-Share* that fully exploits the mobility information of ride requests and taxis for dynamic taxi ridesharing. To address the limitations of existing schemes, *mT-Share* indexes taxis and requests with both spatial partitions and mobility clusters, and optimizes passenger-taxi matching that enables taxis to efficiently serve both online and offline passengers. Experimental results from a large real-world taxi dataset demonstrate that *mT-Share* can response each ride request in milliseconds, and significantly outperforms the state-of-the-art methods, *e.g.*, serving 42% and 62% more ride requests in peak and non-peak hours, respectively.

ACKNOWLEDGMENT

This work was supported in part by China NSFC Grant (No.61802261); NSF Grant of Shenzhen University (No.2018061); and Tencent "Rhinoceros Birds" - Scientific Research Foundation for Young Teachers of Shenzhen University. This work was partially supported by China NSFC Grant (No.61872248, U1736207); Guangdong NSF Grant (No.2017A030312008); Fok Ying-Tong Education Foundation for Young Teachers in the Higher Education Institutions of China (No.161064); and GDUPS (2015). This work was also partially supported by Tianjin Key Laboratory of Advanced Networking (TANK), School of Computer Science and Technology, Tianjin University, Tianjin China, 300350. Kaishun Wu is the corresponding author.

REFERENCES

[1] Gaia initiative. https://outreach.didichuxing.com/research/opendata/.
[2] Openstreetmap. http://www.openstreetmap.org/.
[3] Taxi service research report. http://www.transformcn.com/Topics/2018-08/02/b7944fb3-1b99-4840-89d7-eecaaec67bea.pdf.
[4] R. Baldacci, V. Maniezzo, and A. Mingozzi. An exact method for the car pooling problem based on lagrangean column generation. *Operations Research*, 52(3):422–439, 2004.
[5] X. Bei and S. Zhang. Algorithms for trip-vehicle assignment in ridesharing. In *AAAI*, 2018.
[6] P. S. Castro, D. Zhang, C. Chen, S. Li, and G. Pan. From taxi GPS traces to social and community dynamics: a survey. *ACM Computing Surveys*, 46(2):17, 2013.
[7] L. Chen, Q. Zhong, X. Xiao, Y. Gao, P. Jin, and C. S. Jensen. Price-and-time-aware dynamic ridesharing. In *IEEE ICDE*, 2018.
[8] P. Cheng, H. Xin, and L. Chen. Utility-aware ridesharing on road networks. In *ACM SIGMOD*, 2017.
[9] J.-F. Cordeau and G. Laporte. The dial-a-ride problem (DARP): variants, modeling issues and algorithms. *Quarterly Journal of the Belgian, French and Italian Operations Research Societies*, 1(2):89–101, 2003.
[10] E. W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1(1):269–271, 1959.
[11] P. Goel, L. Kulik, and K. Ramamohanarao. Privacy-aware dynamic ride sharing. *ACM Transactions on Spatial Algorithms and Systems*, 2(1):1–41, 2016.
[12] W. He, K. Hwang, and D. Li. Intelligent carpool routing for urban ridesharing by mining GPS trajectories. *IEEE Transactions on Intelligent Transportation Systems*, 15(5):2286–2296, 2014.
[13] Y. Hou, W. Zhong, L. Su, K. Hulme, A. W. Sadek, and C. Qiao. TASeT: improving the efficiency of electric taxis with transfer-allowed rideshare. *IEEE Transactions on Vehicular Technology*, 65(12):9518–9528, 2016.
[14] 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*, 7(14):2017–2028, 2014.
[15] K. Lin, R. Zhao, Z. Xu, and J. Zhou. Efficient large-scale fleet management via multi-agent deep reinforcement learning. In *ACM KDD*, 2018.
[16] Q. Lin, L. Dengt, J. Sun, and M. Chen. Optimal demand-aware ride-sharing routing. In *IEEE INFOCOM*, 2018.
[17] Z. Liu, Z. Li, M. Li, W. Xing, and D. Lu. Mining road network correlation for traffic estimation via compressive sensing. *IEEE Transactions on Intelligent Transportation Systems*, 17(7):1880–1893, 2016.
[18] Z. Liu, Z. Li, and K. Wu. UniTask: a unified task assignment design for mobile crowdsourcing-based urban sensing. *IEEE Internet of Things Journal*, 6(4):6629–6641, 2019.
[19] Z. Liu, Z. Li, K. Wu, and M. Li. Urban traffic prediction from mobility data using deep learning. *IEEE Network*, 32(4):40–46, 2018.
[20] Z. Liu, P. Zhou, Z. Li, and M. Li. Think like a graph: real-time traffic estimation at city-scale. *IEEE Transactions on Mobile Computing*, 18(10):2446–2459, 2019.
[21] S. Ma, Y. Zheng, and O. Wolfson. T-Share: a large-scale dynamic taxi ridesharing service. In *IEEE ICDE*, 2013.
[22] S. Ma, Y. Zheng, O. Wolfson, et al. Real-time city-scale taxi ridesharing. *IEEE Transactions on Knowledge and Data Engineering*, 27(7):1782–1795, 2015.
[23] Online New York Post news. Apps see surge in riders willing to get comfy with strangers. https://nypost.com/2016/08/13/apps-see-surge-in-riders-willing-to-get-comfy-with-strangers/.
[24] S. Shang, L. Chen, Z. Wei, C. S. Jensen, J.-R. Wen, and P. Kalnis. Collective travel planning in spatial networks. *IEEE Transactions on Knowledge and Data Engineering*, 28(5):1132–1146, 2015.
[25] S. Shang, L. Chen, Z. Wei, C. S. Jensen, K. Zheng, and P. Kalnis. Trajectory similarity join in spatial networks. *Proceedings of the VLDB Endowment*, 10(11):1178–1189, 2017.
[26] S. Shang, L. Chen, Z. Wei, C. S. Jensen, K. Zheng, and P. Kalnis. Parallel trajectory similarity joins in spatial networks. *The VLDB Journal*, 27(3):395–420, 2018.
[27] S. Shang, R. Ding, B. Yuan, K. Xie, K. Zheng, and P. Kalnis. User oriented trajectory search for trip recommendation. In *ACM EDBT*, 2012.
[28] S. Shang, R. Ding, K. Zheng, C. S. Jensen, P. Kalnis, and X. Zhou. Personalized trajectory matching in spatial networks. *The VLDB Journal*, 23(3):449–468, 2014.
[29] N. Ta, G. Li, T. Zhao, J. Feng, H. Ma, and Z. Gong. An efficient ride-sharing framework for maximizing shared route. *IEEE Transactions on Knowledge and Data Engineering*, 30(2):219–233, 2018.
[30] 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 *IEEE ICDE*, 2017.
[31] Y. Tong, Y. Chen, Z. Zhou, L. Chen, J. Wang, Q. Yang, J. Ye, and W. Lv. The simpler the better: a unified approach to predicting original taxi demands based on large-scale online platforms. In *ACM SIGKDD*, 2017.
[32] Y. Tong, L. Wang, Z. Zhou, L. Chen, B. Du, and J. Ye. Dynamic pricing in spatial crowdsourcing: a matching-based approach. In *ACM SIGMOD*, 2018.
[33] Y. Tong, L. Wang, Z. Zhou, B. Ding, L. Chen, J. Ye, and K. Xu. Flexible online task assignment in real-time spatial data. *Proceedings of the VLDB Endowment*, 10(11):1334–1345, 2017.
[34] 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*, 11(11):1633–1646, 2018.
[35] Y. Xu, Y. Tong, Y. Shi, Q. Tao, K. Xu, and W. Li. An efficient insertion operator in dynamic ridesharing services. In *IEEE ICDE*, 2019.
[36] H. Yang, S. C. Wong, and K. I. Wong. Demand–supply equilibrium of taxi services in a network under competition and regulation. *Transportation Research Part B: Methodological*, 36(9):799–819, 2002.
[37] H. Yao, F. Wu, J. Ke, X. Tang, Y. Jia, S. Lu, P. Gong, J. Ye, and Z. Li. Deep multi-view spatial-temporal network for taxi demand prediction. In *AAAI*, 2018.
[38] N. J. Yuan, Y. Zheng, L. Zhang, and X. Xie. T-finder: a recommender system for finding passengers and vacant taxis. *IEEE Transactions on Knowledge and Data Engineering*, 25(10):2390–2403, 2013.
[39] 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*, 12(3):18, 2016.
[40] L. Zhang, T. Hu, Y. Min, G. Wu, J. Zhang, P. Feng, P. Gong, and J. Ye. A taxi order dispatch model based on combinatorial optimization. In *ACM SIGKDD*, 2017.
[41] S. Zhang, Q. Ma, Y. Zhang, K. Liu, T. Zhu, and Y. Liu. QA-share: towards efficient QoS-aware dispatching approach for urban taxi-sharing. In *IEEE SECON*, 2015.
[42] W. Zhang, A. Shemshadi, Q. Z. Sheng, Y. L. Qin, X. Xu, and J. Yang. A user-oriented taxi ridesharing system with large-scale urban GPS sensor data. *IEEE Transactions on Big Data*, 1(1):1–14, 2019.
[43] H. Zheng and J. Wu. Online to offline business: urban taxi dispatching with passenger-driver matching stability. In *IEEE ICDCS*, 2017.
[44] L. Zheng, L. Chen, and J. Ye. Order dispatch in price-aware ridesharing. *Proceedings of the VLDB Endowment*, 11(8):853–865, 2018.
[45] Y. Zheng. Trajectory data mining: an overview. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 6(3):29, 2015.