

**mT-Share**: A Mobility-Aware Dynamic Taxi Ridesharing System

Zhidan Liu, Member, IEEE, Zengyang Gong, Jiangzhou Li, and Kaishun Wu, Member, IEEE,

---

**Abstract**—Due to the wide availability of taxis in a city and tremendous benefits of ridesharing, taxi ridesharing becomes promising and attractive in recent years. Existing taxi ridesharing schemes simply match ride requests and taxis based on partial trip information and omit the offline passengers, who will hail a taxi at roadside without submitting the ride requests to the system. Thus, they are still not efficient and practical. 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 taxis and ride requests 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 makes use of both geographical information and travel directions to index taxis and ride requests, and supports the shortest path based routing and probabilistic routing to serve both online and offline ride requests. In addition, mT-Share proposes a novel payment model to share the ridesharing benefits among the taxi driver and passengers. Extensive evaluations using a large real-world taxi dataset demonstrate the efficiency and effectiveness of mT-Share, which can respond each ride request in milliseconds and be with moderate detour costs and passengers’ waiting time. Compared to state-of-the-art schemes, mT-Share can serve 42% and 62% more ride requests in peak and non-peak hours, respectively. Furthermore, mT-Share can save 8.6% fare for passengers and meanwhile increase 7.8% incomes for taxi drivers, when compared with the regular taxi services.

**Index Terms**—Taxi ridesharing; Mobility pattern; Clustering; Route planning; Payment model

I. INTRODUCTION

Ridesharing utilizes one vehicle to serve multiple passengers, who have similar time schedules and itineraries, and thus potentially brings many benefits for a urban city, e.g., alleviating traffic congestion and reducing energy consumption [29]. Recently, taxi ridesharing becomes promising because of the wide availability of taxis in a city [24], [30], [53]. Different from the carpooling services [16], [36], which normally serve static ride requests with early planned routes, taxi ridesharing is relatively more complex. This is because both taxis and ride requests are extremely dynamic [29]. Once passengers need a ride, they submit their requests immediately without prior planning. Even worse, some passengers prefer to hail a taxi at roadside without explicitly submitting their requests. Compared to private vehicles based ridesharing [5], [11], [12], [36], taxis are operating all the time and thus are more flexible yet complex. These properties cause taxi ridesharing especially challenging, since ride requests have to be assigned to proper taxis timely and the schedule/route of a shared taxi should be constantly updated to guarantee the quality of service [28].

Some valuable efforts have been devoted to design taxi ridesharing schemes [18], [28], [29], [30], [53]. For a given ride request, these schemes usually make use of the request’s origin location and geographical distribution of available taxis to determine a candidate taxi set, and then select the one, which introduces the minimum detour cost while satisfying other passengers’ service requirements, to serve the request. Due to some practical limitations, however, the schemes are not sufficiently efficient yet. First, most previous schemes only utilize partial trip information, i.e., taxis’ current locations and a request’s origin location, to determine the candidate taxi set. As a result, they may not find the most suitable taxi to serve a request. Second, existing schemes merely consider online ride requests, while in practice some passengers may be offline as they hail a taxi at roadside. Thus, such passengers are invisible to the system. Regarding users’ preferences of getting taxi services, a recent taxi service research report shows that 41.68% of users prefer either online booking or offline hailing, while 13.71% of users get taxi services only in an offline manner [3]. As a matter of fact, the amount of offline requests could be quite large (i.e., 13.71% ∼ 55.39% of users), and an appropriate scheme is required to well handle such requests.

In this paper, we will consider a practical taxi ridesharing problem, namely mobility-aware taxi ridesharing (MTR). The MTR problem aims to completely exploit the mobility information from both taxis and ride requests, so as to maximize the number of served requests and minimize the overall detour cost, subject to the constraints of passengers’ deadlines and taxis’ capacities. However, we find that the MTR problem is extremely challenging. The challenges primarily stem from two aspects, namely the high dynamics of online requests and taxis and the uncertainty of offline requests.

To address above problem and improve existing schemes, we propose a novel taxi ridesharing scheme, named mT-Share, by fully exploiting both the known mobility information from ride requests and taxis and the hidden mobility patterns from historical data to match requests with the most suitable taxis. The key idea of mT-Share is that the best passenger-taxi matches should be the pairs of ride requests and taxis, which have geographically close origins and destinations and share the similar travel directions. Therefore, mT-Share proposes the
bipartite map partitioning and mobility clustering to efficiently index ride requests and taxis from these two aspects, respectively. Thanks to the built indexes, \textit{mT-Share} can assign the most suitable taxi to serve a request via the mobility clustering and a proper similarity measure. With the map partitions, \textit{mT-Share} enables two routing modes and further optimizes passenger-taxi matching by improving taxi scheduling efficiency, while simultaneously satisfying the constraints on both requests’ deadlines and taxi’s capacity. By considering the mobility patterns of taxi orders [10], [23], we propose a novel probabilistic routing that allows shared taxis to opportunistically encounter offline requests with much higher probabilities. Furthermore, \textit{mT-Share} proposes a payment model to share the ridesharing benefits among passengers and taxi drivers.

We summarize the contributions of this work as follows.

- We analyze and identify the limitations of previous taxi ridesharing schemes, and further consider the practical MTR problem by exploiting the mobility information to serve both online and offline ride requests.
- We present \textit{mT-Share} to well address the MTR problem. By incorporating the holistic mobility information of ride requests and taxis, \textit{mT-Share} optimizes the indexing of taxis/requests and passenger-taxi matching.
- We further present a novel payment model that is able to fairly share the ridesharing benefits among passengers and drivers, so as to encourage more riders and taxi drivers to join in the ridesharing campaign.
- Extensive experiments have been performed to evaluate \textit{mT-Share} using a large real-world taxi dataset. The results demonstrate that \textit{mT-Share} greatly outperforms the state-of-the-art schemes, e.g., serving 42% and 62% more ride requests in the peak and non-peak hours, respectively. Compared with no taxi ridesharing, \textit{mT-Share} saves 8.6% taxi fare for passengers and meanwhile increases 7.8% incomes for taxi drivers.

The rest of this paper is organized as follows. We review the related works in Section II. The MTR problem is described in Section III. We present the design of \textit{mT-Share} in Section IV, and evaluate the performances in Section V. Finally, we conclude this paper in Section VI.

II. RELATED WORK

Carpooling. Carpooling is also known as recurring ridesharing, which primarily deals with routine commutes, e.g., between home and workplace. As carpooling generally involves a few drivers and riders, it can be solved with linear programming to get the optimal solution [7]. GPS trajectories of users can be leveraged to discover possible carpooling opportunity [16], [37], while \textit{coRide} [50] is proposed to design carpooling service’s schedule and route. Compared to carpooling whose ride requests can be known in a prior, the ride requests of taxi ridesharing are generated instantaneously and the routes of shared taxi are constantly changing. Therefore, taxi ridesharing is more dynamic than carpooling.

Taxi ridesharing can be modelled as a variant of the famous dial-a-ride problem (DARP). In general, DARP aims to design the schedule and route for a number of riders between their origins and destinations [13]. Note that riders in DARP are assumed to specify their pick-up and drop-off locations in advance. Existing works for DARP mainly consider the static scenario [12], where ride requests are pre-known.

Ridesharing. Recently, ridesharing has been widely studied because of its attractive benefits [29]. The dynamic ridesharing can be modeled as a combinatorial optimization problem, and has been proved to be NP-hard [8]. Thus, a variety of heuristic techniques are proposed to optimize the two major stages of ridesharing, i.e., candidate taxi searching [21], [34], [36], [39] and ridesharing routing [19], [42], [57]. For example, Li et al. [21] take both social relations between drivers and riders and the revenue into consideration to select the top-k suitable vehicles for a ride request. Similarly, social preferences are also considered in the passengers-vehicle matching process [34]. Tong et al. [42], [45] have optimized route planning of shared mobility with a smart insertion. In particular, two recent works [44], [49] also make use of demand predictions to plan ridesharing routes, so as to serve more compatible passengers. Our work differs from them by considering both online and offline ride requests, and meanwhile optimizing passenger-taxi matching by fully exploiting the mobility information of ride requests and taxis.

Because of wide availability of taxis in a urban city [10], taxi ridesharing becomes promising and has already attracted substantial research attentions [18], [24], [28], [29], [30], [53]. For example, Zheng et al. develop a mobile-cloud enabled taxi-sharing system called T-Share [29], [30], and Zhang et al. [28] further consider the service quality of taxi ridesharing to improve T-Share. In particular, Hou et al. [18] focus on the transfer-allowed taxi ridesharing using battery limited electric vehicles. Zhang et al. further take the passenger’s acceptance probability on taxi ridesharing into design [53]. Different from our work, these works do not exploit mobility information to achieve appropriate passenger-taxi matching and meanwhile have omitted the offline passengers.

Due to the increasing popularity of ridesharing, other factors of the ridesharing, e.g., the pricing models [11], [21], [41], ridesharing order dispatching [4], [20], [56], partner selection [17], destination matching [31], privacy protection [35], [43], [47], riders’ satisfaction [12], and riders’ attitude on ridesharing [53], have also been studied in the past years.

Taxi demands and dispatching. Taxis play an important role in urban transportation, and thus it is essential to perceive taxi demands and well reposition taxis to balance the supply-demand gap. A great number of works [40], [46], [52] have been made on taxi demand predictions by processing massive historical taxi data. For example, Geng et al. [15] propose a deep learning model to predict the region-level taxi ride-hailing demands. Different from the works which predict demands in regular taxi services, we predict the offline ride requests for taxi ridesharing that is even more challenging. It is worth noting that a recent work [23] presents a ridesharing routing scheme to serve potential riders, which are predicted from the statistics of mobility data. Our scheme differs from it by considering and serving both online and offline requests.

Furthermore, many works [22], [25], [38], [52], [54], [55] have explored the taxi dispatching problem given the known
taxi demands. Specifically, an order dispatching model is proposed to maximize the matching success ratio of ride requests and taxis in [52]. Lin et al. [22] present a fleet management system, which explicitly coordinates idle taxis using the multi-agent deep reinforcement learning theory. Similarly, Tang et al. [38] also apply deep reinforcement learning (DRL) on the order dispatching problem with a deep value-network. Liu et al. [25] improve DRL based taxi dispatching solutions by considering more context information. In addition, Zhao et al. [54] further consider the preference-aware taxi dispatching using online stable matching. These works dispatch a vacant taxi for each individual ride request, while do not consider the ridesharing among passengers.

### III. Problem Statement

In this section, we present the definition, motivation, and the problem statement of mobility-aware taxi ridesharing. We summarize the key notations used in this paper in Table I.

#### A. Preliminary

**Definition 1:** (Road Network) A road network is modelled as a directed graph $G(V, E)$, where a vertex $v \in V$ denotes a geo-location (e.g., road intersection), and an edge $(u, v) \in E$ represents a road segment that owns a weight $cost(u, v)$ to indicate the travel cost from $u$ to $v$.

Specifically, the travel cost is estimated as either a travel time or a travel distance. Once travel speed of a taxi is known, they can be easily converted from one to another. Thus we do not differentiate them throughout the paper and use the travel time or a travel distance. Once travel speed of a taxi is known, they can be easily converted from one to another. Thus we do not differentiate them throughout the paper and use the travel cost.

**Definition 2:** (Ride Request) A ride request is denoted by $r_i = \langle t_{r_i}, o_{r_i}, d_{r_i}, c_{r_i} \rangle$ with a trip origin $o_{r_i} \in V$ and a trip destination $d_{r_i} \in V$. The ride request is released at time $t_{r_i}$ and should be finished before time $c_{r_i}$ by delivering passengers from origin $o_{r_i}$ to destination $d_{r_i}$.

In real-world taxi ridesharing systems, we may adopt two deadlines for pick-up and drop-off, respectively [30]. However, a single deadline for delivery $c_{r_i}$ usually suffices [42]. Given delivery deadline $c_{r_i}$ and travel cost $cost(o_{r_i}, d_{r_i})$ between $o_{r_i}$ to $d_{r_i}$, the pick-up deadline can be calculated as $c_{r_i} - cost(o_{r_i}, d_{r_i})$. The online request will be immediately known once $r_i$ is submitted, but the offline requests could be perceived only when they are encountered by shared taxis. In particular, $r_i$ is used to represent an offline request.

<table>
<thead>
<tr>
<th>Notation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$G(V, E)$</td>
<td>The directed graph of a road network</td>
</tr>
<tr>
<td>$cost(\cdot)$</td>
<td>A function to calculate the travel cost</td>
</tr>
<tr>
<td>$r_i$</td>
<td>The $i$-th ride request</td>
</tr>
<tr>
<td>$d_{r_i}$</td>
<td>The $i$-th online ride request</td>
</tr>
<tr>
<td>$o_{r_i}$</td>
<td>The origin of ride request $r_i$</td>
</tr>
<tr>
<td>$c_{r_i}$</td>
<td>The destination of ride request $r_i$</td>
</tr>
<tr>
<td>$t_j$</td>
<td>Status of the $j$-th taxi</td>
</tr>
<tr>
<td>$S_{t_j}$</td>
<td>Schedule of taxi $t_j$</td>
</tr>
<tr>
<td>$R_{t_j}$</td>
<td>Route of taxi $t_j$</td>
</tr>
<tr>
<td>$\kappa$</td>
<td>Number of partitions for a road network</td>
</tr>
<tr>
<td>$\ell$</td>
<td>A set of map partitions ${P_{t_j}}_{j=1}^{\kappa}$</td>
</tr>
<tr>
<td>$\kappa$</td>
<td>Landmark of partition $P_{t_j}$</td>
</tr>
<tr>
<td>$G_{\ell}(V, E)$</td>
<td>The landmark graph</td>
</tr>
<tr>
<td>$v$</td>
<td>A mobility vector</td>
</tr>
<tr>
<td>$C_{t_j}$</td>
<td>A mobility cluster</td>
</tr>
<tr>
<td>$\theta$</td>
<td>Travel direction difference of two mobility vectors</td>
</tr>
<tr>
<td>$\gamma$</td>
<td>Threshold to determine similar travel direction</td>
</tr>
<tr>
<td>$\gamma_i$</td>
<td>Searching range</td>
</tr>
<tr>
<td>$\ell_v$</td>
<td>Candidate taxi set for ride request $r_i$</td>
</tr>
</tbody>
</table>

**Definition 3:** (Taxi Status) The instantaneous status of the $j$-th taxi is denoted by $t_j = \langle loc_{t_j}, S_{t_j}, R_{t_j} \rangle$, where $loc_{t_j}$ represents taxi $t_j$’s current location, and $S_{t_j}$ and $R_{t_j}$ are the schedule and route of taxi $t_j$, respectively.

**Definition 4:** (Taxi Schedule) A taxi schedule $S_{t_j} = \{s_1, s_2, \cdots, s_m\}$ is a sequence of events for a shared taxi, where each event corresponds to pick-up or drop-off of the ridesharing passenger at some location, e.g., $o_{r_i}$ or $d_{r_i}$, of a ride request $r_1$ and $o_{r_i}$ should appear ahead of $d_{r_i}$.

**Definition 5:** (Taxi Route) A taxi route $R_{t_j}$ is generated for a taxi schedule $S_{t_j}$. It includes the travel path for any two consecutive events in schedule $S_{t_j}$.

Given the request to share a taxi, a valid taxi schedule is established to sequentially pick-up and deliver passengers along a ridesharing route. In previous works [11], [19], [29], [30], [42], travel path between two consecutive event locations is usually set as the shortest path. Therefore, a taxi route can be derived by concatenating a sequence of such shortest paths.

When ridesharing requests are picked up or delivered by taxi $t_j$, both $S_{t_j}$ and $R_{t_j}$ should be timely updated.

#### B. Motivation

The taxi ridesharing problem can be modelled as a combinatorial optimization problem, and has been proved to be NP-hard [8]. Previous works [18], [29], [30], [28], [53] thus have proposed many heuristic techniques to match requests with suitable shared taxis. Generally, they index all taxis and requests using the grids of a road network, and then process each request $r_1$ through the following two major stages.

**Stage 1: taxi searching.** Taxis within a range $\gamma$ around $r_1$’s origin $o_{r_1}$ are chosen as candidate taxis for serving $r_1$.

**Stage 2: ridesharing routing.** The schedule of each candidate taxi $t_j$ is investigated by inserting $r_1$’s pick-up and drop-off events into $S_{t_j}$, subject to passengers’ delivery deadlines and taxi’s capacity. The taxi, which will introduce the minimum cost (e.g., the minimum increased travel cost), is usually selected as the one to serve ride request $r_1$.

In practice, the searching range $\gamma$ could be enlarged [53] and the two stages can be repeated until one proper taxi is
finally found [30]. However, we observe that existing works still suffer from the following two limitations.

- **Inefficient passenger-taxi matching.** Most of existing works determine the candidate taxis with only the given ride request’s origin location [42], [28], [53]. Even though [29], [30] utilize both origin and destination to conduct a dual-side search, such location information are separately considered. Besides, rather than searching for the best taxi to serve a request, some schemes only return the valid taxi that is the first discovered [29], [30]. Relying on such partial trip information, they cannot filter out the invalid taxis at the beginning, and may also miss the best solution, which has the minimum ridesharing cost.

- **Omitting the offline ride requests.** The existing works merely consider online ride requests [18], [28], [29], [30], [53], while in practice there still exist many people who prefer to hail a taxi at roadside without submitting their ride requests [23], [48]. As a concrete example, some elder citizens who are not familiar with online taxi-booking and the passengers who forget to carry out mobile phones are unable to issue online ride requests. According to the statistics on users’ preferences of getting taxi services [23], [32], [46], and thus open a new design space to improve the existing taxi ridesharing schemes.

C. Problem Definition

To improve existing works, we consider a novel **mobility-aware taxi ridesharing (MTR)** problem as follows.

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

- **Capacity constraint:** The number of passengers sharing a taxi cannot exceed the taxi’s capacity at any time;
- **Time constraint:** Passengers of a request should be delivered to their destinations before the specific deadline.

**Challenges.** Different from previous works, the MTR problem considers both online requests and offline requests to improve the efficiency and practicability of taxi ridesharing. Since dynamic ridesharing is NP-hard [8], [12], [42], the MTR problem is challenging to be solved as well, mainly due to the following two challenges.

1) Both ride requests and taxis are quite dynamic, which thus requires that both taxi schedules and routes should be wisely and efficiently planned so as to guarantee the service quality of ridesharing, e.g., minimizing the detour costs.
2) Because the exact information of offline requests cannot be known in advance, it is thus difficult for shared taxis to well serve the offline ride requests. This uncertainty further makes taxi scheduling and routing to be more intricate.

IV. SYSTEM DESIGN

A. Overview

Figure 2 illustrates the framework of $mT$-Share. In general, $mT$-Share takes historical taxi data, real-time taxi statuses and ride requests, and the road map as the input, and arranges the

![Diagram of taxi ridesharing framework](image-url)
available taxis to dynamically serve both online and offline ride requests. On the user side, passengers can either explicitly report their ride requests to mT-Share or hail a shared taxi at roadside in an offline manner. On the taxi side, a shared taxi continuously uploads its status, which includes current location, available seats, and etc., to the server and receives the updated schedule/route from the server.

mT-Share has three major modules, i.e., Taxi/Request Indexing, Passenger-Taxi Matching, and the Payment Model. Specifically, the Taxi/Request Indexing module exploits the mobility patterns that are discovered from historical taxi data to divide the road map into partitions, and classifies taxis and ride requests into mobility clusters based on their travel directions. Both map partitions and mobility clusters are utilized to index and track the shared taxis. With these indexes, the Passenger-Taxi Matching module can effectively search candidate taxis and determine the best one, which introduces the minimum detour cost, to serve a given request. mT-Share supports both basic routing and probabilistic routing, and accelerates them with the partition filtering. In particular, probabilistic routing enables a shared taxi to meet suitable offline requests with a much higher probability. Furthermore, mT-Share proposes a payment model, which determines the taxi fares by sharing the ridesharing benefits between a taxi driver and the passengers.

B. Taxi/Request Indexing

mT-Share will index and track shared taxis and ride requests based on both geographical location and travel direction, which can be achieved by performing bipartite map partitioning and mobility clustering, respectively.

1) Bipartite map partitioning: Rather than dividing a road network graph using geographical information [29], [30] or popular pick-up locations [28], mT-Share classifies the vertices of a road network graph into clusters based on their both geographical locations and transition patterns that are mined from historical taxi data. Specifically, we use k-means clustering algorithm to initially classify all graph vertices into \( k \) transition clusters based on their transition probability vectors. The vertices of a transition cluster will have the similar transition patterns. Empirically, we let \( k_t < \kappa \), and set \( k_t = 20 \) for mT-Share by default.

2) Transition clustering. We regard vector \( \vec{B}_i \) as vertex \( v_i \)'s mobility feature, and use \( k\)-means clustering to group all graph vertices into \( k_t \) transition clusters based on their transition probability vectors. The vertices of a transition cluster will have the similar transition patterns. Empirically, we let \( k_t < \kappa \), and set \( k_t = 20 \) for mT-Share by default.

3) Geo-clustering on transition clusters. For each transition cluster of size \( n \), we group its vertices into \( \left\lfloor \frac{N_k}{N} + \frac{1}{2} \right\rfloor \) spatial clusters based on their locations through \( k\)-means clustering, where \( N = |V| \) denotes the number of all vertices.

Our bipartite map partitioning will repeat above three steps until the \( \kappa \) spatial clusters derived in step 3 do not change. These clusters are regarded as the final partitions of the road network graph \( G \), which is represented as \( P = \{ P_j \}_{j=1}^N \). The vertices of a partition are both geographically close and highly similar on their transition patterns. Such properties facilitate the prediction of suitable offline ride requests by well supporting the probabilistic routing that is introduced later. In Section V-C, we conduct experiments to study the setting of \( \kappa \) and demonstrate the advantage of bipartite map partitioning over the traditional grid-based method on finding more offline passengers. Figure 3(b) demonstrates the result of applying the bipartite map partitioning on the road network graph of Chengdu city, which is shown in Figure 3(a).

For each spatial partition, we calculate its center point as the partition’s landmark. With the spatial partitions and their corresponding landmarks, we construct a landmark graph \( G_\ell \) that will be exploited to speedup route planning.

Definition 7: (Landmark) The landmark of a spatial partition \( P_\ell \) is the vertex \( \ell_\ell \in P_\ell \), which has the minimum distance to all other vertices of partition \( P_\ell \).

Definition 8: (Landmark Graph) A landmark graph is represented as \( G_\ell(\{V_\ell,E_\ell\}) \), where vertices in \( V_\ell \) are the landmarks of all partitions and an edge between two landmarks indicates that their corresponding partitions are adjacent.

Note that the bipartite map partitioning could be periodically executed with a relatively long interval, e.g., one year, when sufficient new taxi data are accumulated to capture the latest transition patterns of passengers in a urban city. Once the map partitions are changed, the corresponding landmarks and the landmark graph should also be accordingly updated.

2) Mobility clustering: Different from existing works that use location information to index taxis and requests only [28], [29], [30], [36], [42], mT-Share builds a mobility vector for
each shared taxi or ride request, and further groups them through mobility clustering based on their travel directions.

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

For each ride request \( r_i \), we use its origin \( o_i \), and destination \( d_i \), to create the mobility vector \( \vec{v}_{r_i} \). For a taxi \( t_j \) that serves \( m \) passengers \((r_1, r_2, \ldots, r_m)\), we take its location \( loc_j \) as the origin of mobility vector \( \vec{v}_{t_j} \), and regard the center of all destinations of the shared passengers \( (i.e., \sum_{m=1}^{m} d_{r_i}) \) as \( \vec{v}_{t_j} \)'s destination. We do not apply mobility clustering for the empty taxis, since they have no fixed travel destinations.

We group taxis and requests into clusters according to their mobility vectors as follows. The first ride request individually forms the initial cluster, and each subsequent request will join an existing cluster or form a new cluster. For each mobility cluster \( C_a \), we maintain a general mobility vector \( \vec{v}_{C_a} \). The origin and destination of \( \vec{v}_{C_a} \) are averaged from the origins and destinations of all cluster members, respectively. Once a new ride request \( r_1 \) arrives, we compare its mobility vector with each general mobility vector. If the travel direction difference between them is sufficiently small, \( r_1 \) will be included into cluster \( C_a \). Specifically, cosine similarity is adopted as the metric to measure the travel direction difference \( \theta \) of two vectors, i.e.,

\[
\cos(\theta) = \frac{\vec{v}_{r_1} \cdot \vec{v}_{C_a}}{|\vec{v}_{r_1}| \times |\vec{v}_{C_a}|}
\]  

When \( \cos(\theta) \geq \lambda \) (where \( \lambda \) is a predefined parameter), \( r_1 \) is considered to travel along a similar direction with these passengers in cluster \( C_a \) and they could share a taxi. Otherwise, \( r_1 \) is forced to form a new mobility cluster. In principle, a smaller \( \lambda \) (i.e., larger \( \theta \)) would increase the ridesharing chances by finding more candidate taxis for a ride request. However, more candidate taxis require more examination time on the taxi schedules and thus prolong response time for each request.

**mT-Share** will update the mobility clusters and their corresponding general mobility vectors only when the ride requests are finished or new requests are received. The updating process introduces negligible computation overheads.

3) **Index of taxis:** mT-Share makes use of both map partitions and mobility clusters to build the index structures, which can facilitate candidate taxi searching for ride requests later.

- **Map partition based indexing.** For each map partition \( P_z \), mT-Share maintains a taxi list \( P_z.L_t \) to record taxi IDs, which are now in or will arrive at map partition \( P_z \) within a time threshold \( T_{mp} \) (e.g., 1 hour). According to their arrival time, these taxi IDs are sorted in an ascending order. The taxi list \( P_z.L_t \) is dynamically updated.  

- **Mobility cluster based indexing.** For each mobility cluster \( C_a \), mT-Share also maintains a taxi list \( C_a.L_t \) to contain taxi IDs, which are now serving requests and meanwhile traveling in a similar direction. The taxi list \( C_a.L_t \) should be updated once mobility cluster \( C_a \) changes.

**Memory complexity.** Based on our indexing structures, each taxi could be indexed by several map partitions and (at most) one mobility cluster, while each request is indexed by only one mobility cluster. Thus, the memory complexity of mT-Share’s indexing overhead is \( O((x + 1)M + R) \), where \( M \) is the number of all available taxis, \( x \) is the number of map partitions a taxi could visit within time threshold \( T_{mp} \), and \( R \) is the number of all ride requests.

C. **Passenger-Taxi Matching**

For each request \( r_1 \), mT-Share will determine its candidate taxis using the indexing structures, and then heuristically investigate all possible taxi schedules to find the most suitable taxi that introduces the minimum ridesharing cost to serve \( r_1 \).

1) **Candidate taxi searching:** Different from existing works, which may increase the searching range gradually and only return one valid (but may not be the best) taxi [28], [29], [30], [53], mT-Share aggressively searches the best taxi to serve each request \( r_1 \). More specifically, the candidate taxi searching range \( \gamma \) for \( r_1 \) is set as the product of a typical taxi driving speed and the waiting time \( \Delta t \) that is estimated as

\[
\Delta t = e_{r_1} - cost(o_{r_1}, d_{r_1}) - t_{r_1},
\]

where \( e_{r_1} = cost(o_{r_1}, d_{r_1}) \) and \( t_{r_1} \) are the pick-up deadline and the release time of \( r_1 \), respectively. From the searching area that centers at \( r_1 \)’s origin \( o_{r_1} \) with radius \( \gamma \), we derive a map partition set \( S_{r_1} \), which intersects with the searching area. For each partition \( P_z \in S_{r_1} \), we get its taxi list \( P_z.L_t \). In addition, by comparing \( r_1 \)’s mobility vector to all existing mobility clusters, we would find a mobility cluster \( C_a \) that shares the similar travel direction with \( r_1 \). Thus, mT-Share determines the candidate taxi set \( T_{r_1} \) for \( r_1 \) as

\[
T_{r_1} = \left\{ \bigcup_{P_z \in S_{r_1}} P_z.L_t \right\} \cap C_a.L_t
\]  

mT-Share further refines set \( T_{r_1} \) using the following rules: (i) including empty taxis \( \{t_j\} \), where \( t_j \in P_z.L_t \) \( (P_z \in S_{r_1}) \) and \( S_{r_1} = \emptyset \); (ii) filtering out the taxis with no idle capacity; (iii) filtering out the taxis, which cannot reach \( r_1 \)’s locating partition \( P_z \) before \( r_1 \)’s pick-up deadline (that can be easily checked from the taxi arrival time recorded in \( P_z.L_t \)). These rules can remove the invalid taxis from candidate taxi set \( T_{r_1} \), so as to save unnecessary computation costs.

2) **Taxi scheduling:** Given set \( T_{r_1} \), taxi scheduling will find the most suitable taxi, which can serve ride request \( r_1 \) while incurring the minimum detour cost. In theory, we should rearrange all events of a taxi schedule \( S_{t_j} \) after incorporating the pick-up event \( o_{r_1} \) and drop-off event \( d_{r_1} \) of \( r_1 \). However, it is prohibited due to the huge computation overheads. Therefore, mT-Share adopts the same design choice as existing works [28], [29], [30], [42], which inserts \( o_{r_1} \) and \( d_{r_1} \) into \( S_{t_j} \) while retaining the existing schedule unchanged. The feasibility of inserting \( r_1 \) into a schedule of taxi \( t_j \) is mainly decided by the time constraints of all requests that are currently served by \( t_j \).

Algorithm 1 presents the taxi scheduling algorithm. For each candidate taxi \( t_j \in T_{r_1} \), we enumerate all taxi schedule instances by inserting event \( o_{r_1} \) and \( d_{r_1} \) into \( S_{t_j} \), where \( o_{r_1} \) is ahead of \( d_{r_1} \). For each schedule instance \( S_{t_j} \) of taxi \( t_j \), we will calculate a route and estimate the detour cost as

\[
detour\ cost = cost(R_{t_j}'') - cost(R_{t_j}'),
\]
Algorithm 1: Taxi Scheduling

```plaintext
1. Input: Ride request \( r_1 \) and candidate taxi set \( T_{r_1} \);
2. Output: A taxi with updated schedule/route for \( r_1 \);
3. foreach taxi \( t_j \in T_{r_1} \) do
   4. foreach schedule instance \( S'_{t_j} \leftarrow \{S_{t_j}, o_{r_1}, d_{r_1}\} \) do
      5. if flag then
         6. \( R'_{t_j} = \text{ProbabilisticRouting}(S'_{t_j}, t_j) \);
      7. else
         8. \( R'_{t_j} = \text{BasicRouting}(S'_{t_j}, t_j) \);
      9. \( \omega = \text{cost}(R'_{t_j}) - \text{cost}(R_{t_j}) \);
   10. Select the taxi schedule instance with the minimum \( \omega \);
```

where \( R'_{t_j} \) is the updated taxi route of \( R_{t_j} \) once taxi \( t_j \) picks up request \( r_1 \). To search the best taxi that incurs the minimum detour cost to serve \( r_1 \), \textit{mT-Share} needs to check all schedule instances of all candidate taxis.

To effectively serve both online and offline requests, \textit{mT-Share} supports two routing modes. A shared taxi normally travels with the basic routing mode. When there are inadequate online requests, the taxi that has sufficient empty seats can enable the probabilistic routing mode to calculate a route to encounter suitable offline requests with a higher probability. Specifically, in Algorithm 1 if indicator flag is true, \textit{mT-Share} invokes function \text{ProbabilisticRouting}() to compute a route for opportunistically seeking offline requests. Otherwise, function \text{BasicRouting}() is used to calculate the shortest path.

\textit{mT-Share} optimizes both functions with \text{PartitionFilter}() by pruning the searching space for fast route planning.

Because route planning usually bottlenecks the efficiency of taxi scheduling [29], [42], \textit{mT-Share} thus optimizes both basic routing and probabilistic routing with a two-phase route planning. Given a taxi schedule instance \( S'_{t_j} \), \textit{mT-Share} calculates the travel path for each consecutive event pair \( (s_z, s_{z+1}) \in S'_{t_j} \) through two phases, \textit{i.e.}, \text{partition filtering} that reduces the searching space of route planning and \text{segment-level routing} that computes the final travel path. By concatenating these travel paths of all consecutive event pairs (\textit{i.e.}, the operation \( \Rightarrow \) in Algorithm 3 and Algorithm 4), \textit{mT-Share} finally derives route \( R'_{t_j} \) for schedule \( S'_{t_j} \). Next, we will detail the two phases.

- Phase 1: Partition filtering

We execute partition filtering on landmark graph \( G_{t_j} \). For any two consecutive events \( (s_z, s_{z+1}) \in S'_{t_j} \), firstly we retrieve their locating partitions \( P_z \) and \( P_{z+1} \), and the corresponding landmarks \( \ell_z \) and \( \ell_{z+1} \). The travel cost between \( \ell_z \) and \( \ell_{z+1} \), \textit{i.e.}, \( \text{cost}(\ell_z, \ell_{z+1}) \), is used to estimate the length of the shortest path between \( s_z \) and \( s_{z+1} \). Furthermore, we generate a mobility vector \( \vec{v}_z \) using their two landmarks. Then, we check each map partition \( P_i \in \mathcal{P} \) using the rules as follows:

- Travel direction rule. The direction difference \( \theta \) between the mobility vector, which points from \( \ell_z \) to \( \ell_{z+1} \), and \( \vec{v}_z \) is small enough, \textit{i.e.}, \( \cos(\theta) \geq \lambda \).
- Travel cost rule. The travel cost of the path, which connects \( \ell_z \) and \( \ell_{z+1} \) via \( \ell_i \), is not obviously larger than travel cost of the shortest path between \( \ell_z \) and \( \ell_{z+1} \), \textit{i.e.},

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

where \( \varepsilon \) is a predefined threshold. We conservatively set \( \varepsilon \) as 1.0 for \textit{mT-Share}.

We retain the map partitions, which satisfy the two rules, into set \( \mathcal{P}_{t_j} \). Algorithm 2 lists the pseudocode of partition filtering. Figure 4(a) illustrates partition filtering applied for events \( (s_z, s_{z+1}) \), where gray partitions are retained into \( \mathcal{P}_{t_j} \).

- Phase 2: Segment-level routing

Instead of planning a route on the original graph \( G \), \textit{mT-Share} builds a subgraph that consists of the vertices and edges from partitions in \( \mathcal{P}_{t_j} \). The subgraph greatly reduces searching space of route planning, and thus \textit{mT-Share} will execute both basic routing and probabilistic routing on this subgraph for better efficiency. We detail the two routing modes as follows.

- Basic routing. For any two consecutive events in a taxi schedule instance \( S_{t_j} \), it will calculate the shortest path. In fact, the basic routing mode has been frequently adopted by existing works for ridesharing routing [28], [29], [30], [42]. Similarly,
mT-Share utilizes the Dijkstra's algorithm [14] to compute the shortest path on the subgraph. Algorithm 3 sketches the pseudocode of basic routing mode.

- **Probabilistic routing.** This mode supports a shared taxi to opportunistically meet some suitable offline ride requests. We say a ride request \( r_i \) is *suitable* only if \( r_i \) travels with the similar direction as the given taxi. Instead of predicting the specified number of offline ride requests, we propose to plan a probabilistic route to maximize the probability of meeting suitable offline ride requests for a given taxi. In principle, we should calculate the probability of meeting suitable requests over each vertex in graph \( G \) and search for the route that can accumulate the maximum probability of meeting suitable ride requests. Such a probabilistic routing has been proved to be NP-Complete [23] and thus is computationally prohibitive.

Instead, mT-Share proposes a heuristic approach to calculate probabilistic routes to avoid huge computations. Specifically, for any two consecutive events \((s_z, s_{z+1})\) in schedule \( S' \), mT-Share exploits transition patterns among partitions to further refine the set \( P_z^+ \). Then, mT-Share builds another much smaller subgraph from the refined partition set for probabilistic route planning. We present the pseudocode of probabilistic routing in Algorithm 4, and detail its main steps as follows.

1. **Probability calculation of suitable requests.** For each partition \( P_i \in P_z^+ \) and a given candidate taxi \( t_j \), we determine a destination partition set \( P_d \) for the suitable requests. For each partition \( P_o \in P \), we build a mobility vector \( \bar{v}_o \) using the landmarks of \( P_i \) and \( P_o \). If the travel direction difference \( \theta \) between \( \bar{v}_a \) and taxi \( t_j \) is sufficiently small (i.e., \( \cos(\theta) \geq \lambda \)), we will retain partition \( P_o \) into set \( P_d \). After obtaining set \( P_d \), we compute a probability \( \pi_z \) of meeting suitable requests within \( P_i \) by summing the transition probability of each vertex in \( P_i \) to each potential destination in set \( P_d \). During bipartite map partitioning, we have already calculated the transition probabilities of any vertex to all partitions (see Section IV-B1). Thus, these calculation results can be reused.

2. **Partition path planning.** We build a landmark graph \( G_z^+ \) using the landmarks and edges of partitions in \( P_z^+ \), where the landmark vertex of partition \( P_i \) is associated with probability \( \pi_z \) as the weight. In general, graph \( G_z^+ \) would be small, we thus is able to enumerate all possible paths, each of which links the landmark of source partition and the landmark of destination partition, to find the maximum weighted path. With landmark vertices of the found path, we can retrieve their corresponding partitions to form a partition path \( H_z^j \), which travels from \( P_z \) to \( P_{z+1} \). At the same time, \( H_z^j \) accumulates the maximum probability at the partition-level.

3. **Fine-grained route planning over partition path.** We construct another weighted graph \( G_z \) using the vertices and edges of partitions in \( H_z^j \), where each vertex \( v_c \) is associated with a weight \( \frac{1}{\pi_z} (\psi_c > 0) \). Specifically, \( \psi_z \) is the accumulated transition probability from vertex \( v_c \) to the destination partition set \( P_d \) of \( v_c \)’s locating partition. We calculate the shortest path with Dijkstra’s algorithm [14] on graph \( G_z \). The derived path should have the minimum weights and meanwhile satisfy the deadline constraints of passengers who share taxi \( t_j \). This path is the final taxi route \( R_z \) that has the highest probability to meet suitable offline requests between \((s_z, s_{z+1})\).

Algorithm 4: Probabilistic Routing

```plaintext
Function ProbabilisticRouting(S, t_j):
    \( R \leftarrow \emptyset; \)
    for \( z = 1 \) to \(|S| - 1\) do
        \( P = \text{PartitionFilter} (s_z, s_{z+1}); \)
        Calculate probability \( \pi_z \) of meeting suitable offline requests for partition \( P_i \in P; \)
        Build weighted landmark subgraph \( G_z^j \) from \( P; \)
        attempt = 0;
        while true do
            Select the maximum weighted path from \( \ell_z \) to \( \ell_z+1 \) on \( G_z^j \) to form the partition path \( H; \)
            Build weighted subgraph \( G_z \) from \( H; \)
            Find the shortest path \( R_z \) using the Dijkstra’s algorithm on \( G_z^j \); \)
            If no valid taxi route is found in step 3, the sub-optimal partition path on \( P_z^+ \) in step 2 is returned. mT-Share will repeat the latter two steps until a valid taxi route (i.e., meeting requests’ delivery deadlines) for event \( s_z \) and \( s_{z+1} \) is found.
            if \( R_z \) is valid then
                break;
            else if attempt > 5 then
                return \emptyset;
            end
            \( R \leftarrow R \bowtie R_z; \)
        end
    end
```

If no valid taxi route is found in step 3, the sub-optimal partition path on \( P_z^+ \) in step 2 is returned. mT-Share will repeat the latter two steps until a valid taxi route (i.e., meeting requests’ delivery deadlines) for event \( s_z \) and \( s_{z+1} \) is found. To avoid endless searching, we set the attempt times as 5. Otherwise, the taxi schedule instance is discarded when there exists no feasible route to link the two events. At last, mT-Share concatenates all the paths of all consecutive event pairs to derive the final taxi route. Figure 4(b) illustrates the partition path and the final path for two consecutive events.

Since probabilistic routing is more computationally expense than the basic routing, a shared taxi \( t_j \) may enable it only when \( t_j \) has sufficient empty seats and there are inadequate online requests. When taxi \( t_j \) encounters an offline ride request \( r_i \), we envision that the taxi driver can report \( r_i \) to the server through some App. Then the server will investigate \( t_j \)’s schedule and route. Taxi \( t_j \) will serve \( r_i \) only if there exists a valid schedule that guarantees the service quality for all requests (including offline request \( r_i \)). Otherwise, the server will quickly dispatch another taxi to serve \( r_i \). The interaction indeed may introduce a slight delay, however, it brings potential benefits for both offline passengers and taxi drivers. Currently, mT-Share plans a probabilistic route to maximize the probability of meeting offline requests given the requests’ delivery deadlines. How to balance the trade-off between this probability and the total detour costs will be explored in our future work.

**Time complexity.** Now we analyze the time complexity of mT-Share’s passenger-taxi matching, which involves the four algorithms. To process each request \( r_i \), mT-Share needs to investigate all possible schedule instances of each candidate taxi in set \( T_{r_i} \) and then chooses the taxi that will introduce the minimum detour cost to serve \( r_i \). If mT-Share adopts basic routing, the time complexity is \( O(|T_{r_i}| m^3 \kappa) \), where \(|T_{r_i}| \) is the size of candidate taxi set \( T_{r_i} \), \( m \) is the number of events in a taxi schedule, and \( \kappa \) is the total number of map partitions.
in \( P \). If \( mT\text{-Share} \) enables the probabilistic routing, the time complexity will be \( O\left(\frac{|P|^2 |N| D |P|}{\kappa} \right) \), where \( D \) is the amount of historical taxi data for calculating transition probabilities, \(|P|\) is the number of partitions retained by Algorithm 2, and \( \kappa \) is the average vertex number of all partitions. Similar as previous studies [12], [42], we assumes the shortest path query will take \( O(1) \) time, because the shortest paths between any two graph vertices could be prepared and cached.

### D. Payment Model

A ridesharing system should bring financial benefits for both ridesharing passengers and taxi drivers, so as to attract more participants. Some payment models have been proposed for the carpooling services [50]. Their involved requests are static and the fares are usually pre-calculated. Thus they cannot work well for dynamic taxi ridesharing. There indeed exist some payment models proposed for dynamic ridesharing [5], [6], [9], [11], [23], [30], [51]. The models in [9], [11], [23] calculate ridesharing fare for each individual passenger with a fixed discounting rate, and do not explicitly consider the benefits for drivers. Although [5], [6], [29], [30], [51] consider both passengers and drivers, their models heavily rely on users' profiles [6] or the complex auction theory [5], [51], and thus cannot be sufficiently flexible and scalable for taxi ridesharing that needs to handle a large number of drivers and passengers. The payment model in [29] even require passengers to pay more if they detour more distances [5], and thus is not fair. Thus, it is necessary to design a payment model that can fairly share ridesharing benefits between taxi drivers and passengers.

The benefits of taxi ridesharing mainly come from the saved travel distance of a ridesharing route when compared to no ridesharing that delivers passengers separately along different routes. Specifically, the benefit \( B \) for taxi ridesharing with \( n \) shared passengers is

\[
B = \sum_{i=1}^{n} f_{sr_i} - \mathcal{F},
\]

where \( f_{sr_i} \) is the regular taxi fare with no ridesharing for ride request \( r_i \), and \( \mathcal{F} \) is the regular taxi fare for a distance equaling to the length of ridesharing route. The total fare of \( n \) passengers without ridesharing is given by \( \sum_{i=1}^{n} f_{sr_i} \), while the taxi ridesharing fare is \( \mathcal{F} \). Their difference is thus the ridesharing benefit \( B \). \( mT\text{-Share} \) partitions the benefit between taxi driver and all passengers (as a group) with a rate \( \beta \). A driver will obtain benefit \((1 - \beta) \times B\), while all ridesharing passengers share the benefit \( \beta \times B \).

\( mT\text{-Share} \) splits the benefit among passengers proportionally according to their detour rates. Specifically, the detour rate is defined as the ratio between the detour distance and the shortest path length. For ride request \( r_i \) that arrives at the destination, the detour rate is

\[
\sigma_i = \eta + \frac{\text{cost}(\mathcal{R}_{r_i}) - \text{cost}(\mathcal{R}_{sr_i})}{\text{cost}(\mathcal{R}_{sr_i})},
\]

where \( \mathcal{R}_{r_i} \) is the shared route \( r_i \) has traveled, \( \mathcal{R}_{sr_i} \) is the shortest path for request \( r_i \), and \( \eta \) is the base rate. Note that base rate \( \eta \) is introduced to guarantee that all passengers can gain benefits from the ridesharing even they do not take any detour, e.g., all ride requests have the same pick-up and drop-off locations. For a ride request \( r_j \) that has not been completed yet, the detour rate is assumed as

\[
\sigma_j = \eta + \frac{\text{cost}(\mathcal{R}_{r_j}) + \text{cost}(\mathcal{R}_{sr_j}) - \text{cost}(\mathcal{R}_{sr_j})}{\text{cost}(\mathcal{R}_{sr_j})},
\]

where \( \mathcal{R}_{r_j} \) is the shared route \( r_j \) has already traveled, and we assume the taxi will deliver \( r_j \) with the shortest path for the remaining trip, i.e., \( \mathcal{R}_{sr_j} \) denotes the shortest path from \( r_j \)'s destination to \( r_j \)'s destination.

Therefore, for ride request \( r_i \) that arrives at the destination now, the taxi ridesharing benefit will be \( \beta \times B \times \frac{\sigma_i}{\sum_{z=1}^{N} \sigma_z} \), and thus the taxi ridesharing fare is

\[
f_{r_i} = f_{sr_i} - \beta \times B \times \frac{\sigma_i}{\sum_{z=1}^{N} \sigma_z}.
\]

From this payment model, we see a passenger will not pay more than the regular taxi service and a taxi driver can earn more from the ridesharing. In addition, passengers with relatively large detour rates can receive more compensations. Therefore, the payment model will largely encourage more passengers and taxi drivers to participate in taxi ridesharing.

### V. PERFORMANCE EVALUATION

#### A. Experimental Setup

**Dataset.** We perform extensive experiments with a large taxi dataset that is publicly released by Didi’s GAIA initiative [1]. The dataset totally has 7065907 taxi transactions, which were collected within the 2\textsuperscript{nd} Ring Road from Chengdu city, China, in 2016. Specifically, each transaction includes a transaction ID, a taxi ID, and a ride request, which consists of the release time, the pick-up latitude/longitude, and the drop-off latitude/longitude. We employ the data from two specific time periods to simulate two representative ridesharing scenarios:

- **Peak scenario.** In general, taxis have to handle a large number of online requests during the peak hours. Thus, the offline requests are ignored in this scenario. The data from 8:00AM-9:00AM of a typical workday that has the most hourly requests, i.e., 29534, is used to evaluate \( mT\text{-Share} \) in the peak hours.

- **Non-peak scenario.** During non-peak hours, most taxis have sufficient idle capacity while there exist inadequate online requests. Thus, taxi drivers can exploit probabilistic routing to seek offline passengers. Here we assume a taxi with half of the capacity in idle will enable the probabilistic routing. The data from 10:00AM-11:00AM of a typical weekend, i.e., 15480 rides requests, is used to evaluate \( mT\text{-Share} \) in the non-peak hours. Besides, we randomly pick 5000 requests out of all and make their release time and origin/destination to be invisible to the system. These requests are viewed as the offline requests.

We use the rest taxi data for bipartite map partitioning and probability calculations of meeting suitable requests. Statistics about the taxi data are shown in Figure 5. Figure 5(a) presents the average taxi utilization in workdays and weekends. The taxi utilization is defined as the proportion of serving requests...
within each hour. From Figure 5(a), we see that the utilization ratios of 8:00AM-9:00AM in workdays and 10:00AM-
11:00AM in weekends are 56% and 41%, respectively. Figure 5(b) further shows the travel time distribution of all taxi
trips in the dataset, e.g., the 90-percentile and 50-percentile trip travel time are 30 minutes and 15 minutes, respectively.

We download road network data within the 2nd Ring Road of Chengdu city from OpenStreetMap [2] and model the road
network as a directed graph \( G(V,E) \), which contains 214440 vertices and 406330 edges in total, covering an area of more
than 70 km². The testing road network is shown in Figure 3(a).

Compared schemes. We compared mT-Share against the following four baseline schemes.

1. No-Sharing operates as the regular taxi service with no ridesharing at all. It assigns a ride request to the geographically
nearest idle taxi within the searching range \( \gamma \).

2. T-Share [29], [30], one of the state-of-the-art schemes, indexes all requests and taxis using grids and selects candidate
taxis by conducting a dual-side search from both origin and destination of a request with the searching range \( \gamma \). However,
it only returns the first valid candidate rather than the best one.

3. pGreedyDP [42], one of the state-of-the-art schemes, indexes all requests and taxis using grids like T-Share, and selects candidate taxis within the searching range \( \gamma \) around the request \( r_i \)'s origin. To improve the taxi scheduling efficiency, it determines the event insertions of \( r_i \) into an existing schedule through dynamic programming.

4. mT-Sharepro is the version of mT-Share with enabled probabilistic routing. Since it will introduce huge computation
costs, mT-Sharepro is only evaluated in the non-peak scenario. It is desirable for taxi drivers to serve offline requests during such periods that have inadequate online requests [10], [48].

For fair comparisons, we adjust the settings of T-Share and pGreedyDP, and enable them to serve offline requests as well.
Along the taxi route provided by T-Share or pGreedyDP, if a taxi \( t_j \) that has sufficient empty seats happens to meet suitable offline requests \( r_i \), which can be validly inserted into \( t_j \)'s schedule, then taxi \( t_j \) could serve request \( r_i \). Similarly, they serve offline ride requests only in non-peak scenario as well.

Performance metrics. We evaluate all the schemes on the following performance metrics.

- Number of served requests is the number of ride requests, which have been timely served.
- Response time is the processing time for a ridesharing scheme to match a suitable taxi with the request.
Comparisons in the peak scenario. Figure 6 shows that the four schemes can serve more requests when there are more available taxis. With No-Sharing, a taxi can accomplish about 2 ride requests on average within one peak hour. Compared to No-Sharing, taxi ridesharing can indeed serve much more requests. From Figure 6, we find that pGreedyDP outperforms T-Share because it has optimized the ridesharing routing and thus is able to find better passenger-taxi matches. Among all the schemes, mT-Share serves the most requests since mT-Share has optimized both candidate taxi searching and passenger-taxi matching by efficiently exploiting the mobility information. Taking the case of 3000 taxis as an example, No-Sharing, T-Share, pGreedyDP, and mT-Share have served 6534, 8441, 8868, and 11906 ride requests, respectively. Compared to T-Share and pGreedyDP, i.e., the state-of-the-art works, mT-Share can serve 42% and 36% more ride requests, respectively.

When there are more available taxis, the response time of all schemes increases, as shown in Figure 7. No-Sharing can respond a request within 1 millisecond (ms). mT-Share will take a bit more time to process a request than T-Share, and pGreedyDP has the largest response time. It is possibly because pGreedyDP spends much time to determine the low bound detour cost for each candidate taxi. In general, mT-Share responds a ride request within 35 ~ 140 ms, and outperforms pGreedyDP by 4 ~ 10 times on the metric of response time.

To better understand the results in Figure 6 and Figure 7, we present the average numbers of candidate taxis for a request for different schemes in Table III. With the same searching range $\gamma$, No-Sharing has the smallest candidate taxi set since it only considers vacant taxis. T-Share has much fewer candidate taxis than pGreedyDP and mT-Share because its dual-side search mistakenly removes many possible taxis [42]. pGreedyDP has the most candidate taxis among the four schemes. According to the candidate taxi searching strategy in Section IV-C, mT-Share will aggressively consider all possible candidate taxis in the partitions that intersect with the searching range, and filter out invalid candidates by comparing the travel directions of a taxi and the request. As a result, mT-Share can initially remove many invalid candidate taxis while preserving possible ones. These are the reasons why mT-Share can respond each request quickly while serving the most requests.

Figure 8 reports the detour time. No-Sharing introduces no detour, while the ridesharing schemes have detour time of 1 ~ 4 minutes. More taxis potentially allow all ridesharing schemes to search a more suitable taxi for a request, which thus reduces the detour time. In general, T-Share has the minimum detour time, while mT-Share is quite close to T-Share and holds the second place. However, pGreedyDP nearly doubles the detour time of T-Share. Compared to pGreedyDP, mT-Share improves the average detour time by 31% to 40%.

We study the waiting time of all schemes and report the results in Figure 9. In general, more taxis potentially allow each scheme to find a nearby taxi to serve each request, and thus the waiting time can be reduced. With no ridesharing, No-Sharing has the minimum taxi supplies among all schemes given the same number of taxis. Thus it has a relatively larger waiting time around 1 minute. T-Share has the smallest waiting time since it usually returns the nearest vacant taxi. Both mT-Share and pGreedyDP try to maximize the number of served requests and minimize the total detour costs, they thus have large waiting time. mT-Share has a bit longer waiting time than pGreedyDP, while the gap is quite small, i.e., < 0.5 minutes.

Comparisons in the non-peak scenario. The number of served requests in non-peak scenario is shown in Figure 10. By comparing with Figure 6, we observe that the advantage of ridesharing over No-Sharing is diminishing. This is possibly because there are much fewer requests in the non-peak hours. We even see that T-Share has similar performances as No-Sharing on the number of served requests in some settings. mT-Share and mT-Sharepro still serve much more requests than T-Share and pGreedyDP. The probabilistic routing indeed helps mT-Sharepro to serve more requests, e.g., improving mT-Share by 13% ~ 24%. Compared to T-Share and pGreedyDP, i.e., state-of-the-art schemes, mT-Sharepro can serve 62% and 58% more requests, respectively.

Figure 11 shows the response time of all schemes in non-peak scenario. Comparing to their results in Figure 7, we find No-Sharing, T-Share, pGreedyDP, and mT-Share have quite similar performances both scenarios. As probabilistic routing involves huge computation costs for finding the route with the highest probability of meeting suitable requests, the response time of mT-Sharepro is thus much greater than mT-Share, i.e., 2.5 ~ 4.5 times slower. The performance gap between mT-Sharepro and pGreedyDP becomes smaller when there are
more taxis. However, mT-Sharepro still responds a ride request much faster than pGreedyDP with 81% ~ 497% performance gain. Figure 7 and Figure 11 also demonstrate that our scheme has a good scalability and can respond each request quickly.

Since the route planning algorithms of No-Sharing, T-Share, pGreedyDP, and mT-Share are the same in both scenarios, their detour time results in non-peak scenario as shown in Figure 12 are quite similar with the results in peak scenario in Figure 8. Because probabilistic routing may return some long taxi routes to encounter offline passengers by chance, mT-Sharepro has the largest detour time among all schemes. Even so, the detour time difference between mT-Sharepro and pGreedyDP is still small, i.e., ≤ 0.5 minutes. It implies that mT-Sharepro greatly improves pGreedyDP at a minor cost.

We present the waiting time of all schemes in Figure 13. We observe an obvious decrease in trend of waiting time when the number of available taxis is increased. Compared to the results in Figure 9, the waiting times of the five schemes in non-peak scenario become larger. This is because we have fewer requests in the non-peak hours, and a taxi usually needs to travel more distances to pick up the passengers. Because of the probabilistic routing, mT-Sharepro has the largest waiting time, e.g., 2 minutes longer than pGreedyDP.

**Memory overhead.** The ridesharing schemes usually index both taxis and ride requests to accelerate the passenger-taxi matching, which consumes system memory. We thus compared their memory overheads with 3000 taxis in the peak scenario, which indicates the upper bound of memory costs. It is worth noting that mT-Share and mT-Sharepro have the same memory costs. Table IV presents the statistic results. Compared to T-Share and pGreedyDP, mT-Share builds indexes with both map partitions and mobility clusters, and thus mT-Share has about 39.5% and 38.7% larger indexes, respectively. As a result, mT-Share consumes 16.0% and 40.7% more memories than T-Share and pGreedyDP, respectively. Fortunately, such memory overheads are negligible since current servers are equipped with sufficient memory.

**C. Detailed Evaluation**

Next we will conduct experiments to explore the impacts of important parameters and alternative designs for mT-Share.

**Impact of partition number κ.** In this experiment, we vary the partition number κ while keeping other settings in peak scenario. As shown in Figure 14(a), for the three schemes the number of served requests increases at the beginning and then decreases beyond κ = 150. The κ values, either too small or oversize, will lead to fewer candidate taxis, and thus influence ridesharing performances. When κ is varied from 50 to 150, the average size of candidate taxi sets increases by 17% and meanwhile the number of served requests increases from 8234 to 8753. In contrary, more partitions lead to a shrinking candidate taxi set, and the served ride requests are reduced.

**Impact of capacity.** Except varying the taxi capacity, we conduct this experiment using default settings as well in the peak scenario. When the taxi capacity is enlarged, the same number of available taxis will have much more supplies and thus they could serve more ride requests. As shown in Figure 14(b), the larger taxi capacity brings more served ride requests. Specifically, compared to capacity = 2, mT-Share serves 12% more requests when capacity = 6.

**Impact of map partitioning strategies.** Rather than using grids on the road network graph as previous works [29], [30],

---

**TABLE IV**

<table>
<thead>
<tr>
<th>Metric</th>
<th>T-Share</th>
<th>pGreedyDP</th>
<th>mT-Share</th>
<th>mT-Sharepro</th>
</tr>
</thead>
<tbody>
<tr>
<td>Index Size (Bytes)</td>
<td>34404</td>
<td>34617</td>
<td>48018</td>
<td></td>
</tr>
<tr>
<td>Overall Memory (MB)</td>
<td>381</td>
<td>314</td>
<td>442</td>
<td></td>
</tr>
</tbody>
</table>

---

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/JIOT.2021.3102638, IEEE Internet of Things Journal.
[39], [42], mT-Share instead adopts a bipartite map partitioning strategy, which considers both geographical information and mobility patterns. We compare the performances of mT-Share with different partitioning strategies in both scenarios with the default settings, and present the results in Table V. We see bipartite map partitioning indeed improves the performances in both scenarios. Specifically, bipartite map partitioning improves the number of served requests by 6% at least, while reducing the detour time by 3% - 7%. The results prove that mobility-aware map partitioning can benefit ridesharing.

**Impact of searching range γ.** All schemes determine a candidate taxi set for each request with a searching range γ, and we study its impacts on the detour time and waiting time in peak scenario. Figure 15 shows that a larger searching range γ usually leads to more both detour time and waiting time. No-Sharing has no detour at all. Typically, ridesharing schemes will find more candidate taxis in a larger searching range, and the selected taxi may be farther from the requests and meanwhile has more detour cost. The sum of detour time and waiting time can implicitly represent the service quality of a taxi ridesharing scheme, where a larger value indicates that the passengers need to spend more extra time for the taxi trip. Figure 15 reports that T-Share wins the best service quality and mT-Share has better service quality than pGreedyDP.

**Impact of routing schemes.** We study whether probabilistic routing benefits for finding offline requests. In this experiment, we combine basic routing or probabilistic routing with T-Share, pGreedyDP, or mT-Share, and then run each combinatorial scheme in non-peak scenario with default settings. Figure 16 illustrates the compositions of served requests for different combinations. From Figure 16(a), although the basic routing based schemes may encounter a few offline passengers by chance, probabilistic routing indeed enlarges the probability by serving more offline requests, as shown in Figure 16(b). By comparing Figure 16(a) and 16(b), we find that probabilistic routing brings 89%, 46%, and 34% more offline requests for T-Share, pGreedyDP, and mT-Share, respectively. By exploiting the mobility patterns for effective route planning, overall they serve 26%, 17%, and 14% more requests, respectively.

**Impact of flexible factor ρ.** We perform experiments to explore the impact of the flexible factor ρ in the peak scenario with default settings. Firstly, we study the impact of ρ on the passengers’ waiting time and plot the results in Figure 17. As ρ does not influence No-Sharing, we thus omit it. A larger ρ essentially indicates that passengers can tolerant more detour time, and thus much farther taxi may be selected to serve them, resulting in longer waiting time. Generally, T-Share has the shortest waiting time, and mT-Share has relatively longer waiting time. The performance gap between pGreedyDP and mT-Share is quite small, i.e., < 1.2 minutes.

Figure 18 shows that when we increase ρ, the detour time also increases. The shared taxis can serve more requests with a larger ρ, but the number of served requests slightly increases beyond ρ = 1.3. It implies that more detour time brings about negligible benefit when we choose even larger ρ. For example, the number of served requests and detour time are 8753 and 2.5 minutes, respectively, when ρ = 1.3. The numbers increase to 9140 and 3.6 minutes, respectively, when ρ = 1.4. It means that 4% improvement on served requests comes at the expense of 48% increase of detour time.

We study the monetary benefits of ridesharing for both passengers and taxi drivers based on our payment model. Figure 19 shows that taxi ridesharing indeed has economic advantages. Specifically, a larger flexible factor ρ will save more fares for ridesharing passengers while the profit for taxi drivers is decreasing. This is because larger ρ cannot lead to remarkable increase of served requests (as reported in Figure 18) while the travel distance grows greatly. As a result, the benefit is reduced for drivers. Specifically, the passengers can save 8.6% taxi fare while the drivers can obtain 7.8% more incomes when ρ = 1.3, which is a good setting for both sides.

**Impact of threshold λ.** We study the impact of parameter λ in mobility clustering by varying the maximum travel direction difference θ from 30° to 75°. The experiment is performed in the peak scenario with default settings. Figure 20 shows that
when we increase $\theta$ (i.e., decreasing $\lambda$), the number of served requests increases slightly while the response time increases greatly. This is because a smaller $\lambda$ brings more candidate taxis for each request and thus enhances the ridesharing chances. However, it also introduces huge computation costs to examine the possible taxi schedules. Therefore, we adopt $\lambda = 0.707$ (i.e., $\theta = 45^\circ$) to balance the number of served requests and the response time for a better system performance.

**Impact of used data amounts.** To investigate the scalability of our system, we make use of the taxi data during 7:00AM-20:00PM of one typical workday and one typical weekend for experiments. We run $mT$-Share and $mT$-Sharepro for workday data and weekend data, respectively. For each setting of experiments on weekend data, we assume one third of all requests as the offline. We gradually increase the hours of used taxi data, and present the results in Figure 21. As shown in Figure 21(a), the total execution time raises linearly, when we increase the data amounts for both workday and weekend. For all the 13 hours of taxi data, $mT$-Share can complete all calculations within 4 hours for workday data. Although probabilistic routing takes time, $mT$-Sharepro can still finish within 6 hours for weekend data. Figure 21(b) shows that the response time is quite stable in both workday and weekend. Specifically, the average response time for workday and weekend are 110 ms and 420 ms, respectively. These results prove that our scheme is computationally efficient and can well scale to large amounts of data for a large city.

**VI. CONCLUSION**

This paper presents a novel dynamic taxi ridesharing scheme – $mT$-Share, which fully exploits the mobility information of taxis and ride requests. To improve the existing works, $mT$-Share utilizes both map partitions and mobility clusters to index taxis and ride requests, and has optimized passenger-taxi matching with efficient routing. In particular, $mT$-Share supports the shared taxis to well serve both online and offline ride requests. Based on a large real-world taxi dataset, our experimental evaluations show that $mT$-Share can respond each request within milliseconds, and greatly outperforms the state-of-the-art works, e.g., serving 42% and 62% more ride requests in peak and non-peak hours, respectively.

**ACKNOWLEDGMENT**

This work was supported in part by the National Science Foundation of China (NSFC) under Grant No.61802261, No.61872248, and No.U2001207. This work was also partially supported by the grant of Guangdong Basic and Applied Basic Research Foundation (No.2020A1515011502 and No.2017A030312008), Shenzhen Science and Technology Foundation (No.ZDSYS20190902092853047, R2020A045), the Project of DEGP (No.2019KCXTD005), the Guangdong “Pearl River Talent Recruitment Program” under Grant 2019ZT08X603. Kaishun Wu is the corresponding author.

**REFERENCES**
