Real-Time City-Scale Taxi Ridesharing

Shuo Ma, Yu Zheng, *Senior Member, IEEE*, and Ouri Wolfson, *Fellow, IEEE*

Abstract—We proposed and developed a taxi-sharing system that accepts taxi passengers' *real-time* ride requests sent from smartphones and schedules proper taxis to pick up them via ridesharing, subject to time, capacity, and monetary constraints. The monetary constraints provide incentives for both passengers and taxi drivers: passengers will not pay more compared with no ridesharing and get compensated if their travel time is lengthened due to ridesharing; taxi drivers will make money for all the detour distance due to ridesharing. While such a system is of significant social and environmental benefit, e.g., saving energy consumption and satisfying people's commute, real-time taxi-sharing has not been well studied yet. To this end, we devise a mobile-cloud architecture based taxi-sharing system. Taxi riders and taxi drivers use the taxi-sharing service provided by the system via a smart phone App. The Cloud first finds candidate taxis quickly for a taxi ride request using a taxi searching algorithm supported by a spatio-temporal index. A scheduling process is then performed in the cloud to select a taxi that satisfies the request with minimum increase in travel distance. We built an experimental platform using the GPS trajectories generated by over 33,000 taxis over a period of three months. A ride request generator is developed (available at http://cs.uic.edu/~sma/ridesharing) in terms of the stochastic process modelling real ride requests learned from the data set. Tested on this platform with extensive experiments, our proposed system demonstrated its efficiency, effectiveness and scalability. For example, when the ratio of the number of ride requests to the number of taxis is 6, our proposed system serves three times as many taxi riders as that when no ridesharing is performed while saving 11 percent in total travel distance and 7 percent taxi fare per rider.

Index Terms—Spatial databases and GIS, taxi-sharing, GPS trajectory, ridesharing, urban computing, intelliegent transportation systems

1 Introduction

Taxi is an important transportation mode between public and private transportations, delivering millions of passengers to different locations in urban areas. However, taxi demands are usually much higher than the number of taxis in peak hours of major cities, resulting in that many people spend a long time on roadsides before getting a taxi. Increasing the number of taxis seems an obvious solution. But it brings some negative effects, e.g., causing additional traffic on the road surface and more energy consumption, and decreasing taxi driver's income (considering that demands of taxis would be lower than number of taxis during off-peak hours).

To address this issue, we propose a taxi-sharing system that accepts taxi passengers' real-time ride requests sent from smartphones and schedules proper taxis to pick up them via taxi-sharing with time, capacity, and monetary constraints (the monetary constraints guarantee that passengers pay less and drivers earn more compared with no taxi-sharing is used). Our system saves energy consumption and eases traffic congestion while enhancing the capacity of commuting by taxis. Meanwhile, it reduces the taxi fare of taxi riders and increases the profit of taxi drivers.

Unfortunately, real-time taxi-sharing has not been well explored, though ridesharing based on private cars, often known as carpooling or recurring ridesharing, was studied for years to deal with people's routine commutes, e.g., from home to work [1], [2]. In contrast to existing ridesharing, real-time taxi-sharing is more challenging because both ride requests and positions of taxis are highly dynamic and difficult to predict. First, passengers are often lazy to plan a taxi trip in advance, and usually submit a ride request shortly before the departure. Second, a taxi constantly travels on roads, picking up and dropping off passengers. Its destination depends on that of passengers, while passengers could go anywhere in a city.

In this paper, we report on a system based on the mobile-cloud architecture, which enables real-time taxi-sharing in a practical setting. In the system, taxi drivers independently determine when to join and leave the service using an App installed on their smartphones. Passengers submit real-time ride requests using the same App (if they are willing to share the ride with others). Each ride request consists of the origin and destination of the trip, time windows constraining when the passengers want to be picked up and dropped off (in most case, the pickup time is present). On receiving a new request, the Cloud will first search for the taxi which minimizes the travel distance increased for the ride request and satisfies both the new request and the trips of existing passengers who are already assigned to the taxi, subject to time, capacity, and monetary constraints. Then the existing passengers assigned to the taxi will be inquired by the cloud whether they agree to pick up the new passenger given the possible decrease in fare and increase in travel time. Only with a unanimous agreement, the updated schedules will be then given to the corresponding taxi drivers and passengers.

We place our problem in a practical setting by exploiting a real city road network and the enormous historical taxi trajectory data. Compared to existing carpooling systems, our proposed ridesharing model considers more practical

• *S. Ma and O. Wolfson are with the Computer Science Department, University of Illinois at Chicago, Chicago, IL 60607.
E-mail: {sma, wolfson}@cs.uic.edu.*
• *Y. Zheng is with Microsoft Research, Beijing 100080, P.R, China.
E-mail: yuzheng@microsoft.com.*

*Manuscript received 28 Oct. 2013; revised 11 June 2014; accepted 11 June 2014. Date of publication 30 June 2014; date of current version 1 June 2015.
For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.
Digital Object Identifier no. 10.1109/TKDE.2014.2334313*

constraints which include time windows, capacity, and monetary constraints for taxi trips. In addition, our work proposes efficient searching and scheduling algorithms that are capable of allocating the "right" taxi among tens of thousands of taxis for a query in milliseconds. The contribution of this paper is many fold:

- We proposed and developed a taxi-sharing system using the mobile-cloud architecture. The cloud integrates multiple important components including taxi indexing, searching, and scheduling. Specifically, we propose a spatio-temporal indexing structure, a taxi searching algorithm, and a scheduling algorithm. Supported by the index, the two algorithms quickly serve a large number of real-time ride requests while reducing the travel distance of taxis compared with the case without taxi-sharing.
- Our paper considers and models monetary constraints in ridesharing. These constraints provide incentives not only for passengers but also for taxi drivers: passengers will not pay more compared with no ridesharing and get compensated if their travel time is lengthened due to ridesharing; taxi drivers will make money for all the reroute distance due to ridesharing. The monetary constraints makes our modeling of the taxi ridesharing problem more realistic.
- We performed extensive experiments to validate the effectiveness of taxi-sharing as well as the proposed system's efficiency and scalability. According to the experimental results, the fraction of ride requests that get satisfied is significantly increased by three times meanwhile riders save 7 percent in taxi fare via taxi-sharing when the taxis are in high demand. Furthermore 2 million liter of gasoline can be saved each year in Beijing by taxis alone if taxi-sharing is allowed.

The rest of this paper is organized as follows. In Section 2, we formally describe the real-time taxi-sharing problem. Section 3 overviews the architecture of our proposed system. Section 4 describes the index of taxis used and the taxi searching algorithm. Section 5 describes the scheduling process. We present the evaluation in Section 6 and summarize the related work in Section 7.

Compared with our earlier work [3], this paper claims following contributions. First, we devised and implemented a cloud-mobile based taxi-sharing system, where the mobile App for taxi riders and drivers are designed. The detailed interactions among the mobile Apps of taxi riders, drivers, and the cloud are presented in Section 3. Second, we refined our taxi-sharing model by introducing the monetary constraints for both drivers and riders in Section 5.2. This is one crucial step towards a practical deployment of such a system. In addition, we discussed the time complexity of reordering the pickup and drop-off locations in the scheduled route of a taxi at the beginning of Section 5 and showed that this step is not necessary in practice via experiments in Section 6.2.3. Third, we conducted more comprehensive experiments to validate our system. For example, in Section 6.1.5, we introduce new measurements, such as *Taxi-sharing Rate* and *Seat Occupancy Rate*, to evaluate the effectiveness of ridesharing. We also test the performance of our approach by changing the parameter of the monetary constraints in Section 6.2.1.

2 The Real-Time Taxi-Sharing Problem

The real-time taxi-sharing problem consists of a data model, constraints, and an objective function. We describe each part separately below before giving the formal definition of the problem.

2.1 Data Model

2.1.1 Ride Request

A *ride request* Q is associated with a timestamp $Q.t$ indicating when Q was submitted, a origin point $Q.o$, a destination point $Q.d$, a time window $Q.pw$ defining the time interval when the rider wants to be picked up at the origin point, and a time window $Q.dw$ defining the time interval when the rider wants to be dropped off at the destination point. The early and late bounds of the pickup window are denoted by $Q.pw.e$ and $Q.pw.l$ respectively. Likewise, $Q.dw.e$ and $Q.dw.l$ stand for that of the delivery window.

In practice, a rider only needs to explicitly indicate $Q.d$ and $Q.dw.l$, as most information of a ride request can be automatically obtained from a rider's mobile phone, e.g., $Q.o$ and $Q.t$. In addition, we can assume that both $Q.pw.e$ and $Q.dw.e$ equals to $Q.t$, and $Q.pw.l$ can be easily obtained by adding a fixed value, e.g., 5 minutes, to $Q.pw.e$.

2.1.2 Taxi Status

A taxi status V represents an instantaneous state of a taxi and is characterized by the following fields.

- $V.ID$. The unique identifier of the taxi.
- $V.t$. The time stamp associated with the status.
- $V.l$. The geographical location of the taxi at $V.t$.
- $V.s$. The current schedule of V, which is a temporally-ordered sequence of origin and destination points of n ride requests $Q_1, Q_2, \dots Q_n$ such that for every ride request Q_i, $i = 1, \dots, n$, either 1) $Q_i.o$ precedes $Q_i.d$ in the sequence (referred to as the precedence rule thereafter), or 2) only $Q_i.d$ exists in the sequence.
- $V.r$. The current projected route of V, which is a sequence of road network nodes calculated based on $V.s$.

From the definition, it is clear that the schedule of a vehicle status is dynamic, i.e., changes over time. For example, a schedule involving two ride requests Q_1 and Q_2 could be $Q_1.o \rightarrow Q_2.o \rightarrow Q_1.d \rightarrow Q_2.d$ at a certain time. The schedule is updated to $Q_2.o \rightarrow Q_1.d \rightarrow Q_2.d$ once the taxi has passed point $Q_1.o$.

2.2 Constraints

The crux of the taxi-sharing problem is to dispatch taxis to ride requests, subject to certain constraints. We say that a taxi status V satisfies a ride request Q or Q is *satisfied* by V if the following constraints are met.

- *Vehicle capacity constraint*. The number of riders that sit in the taxi does not exceed the number of seats of a taxi at any time.

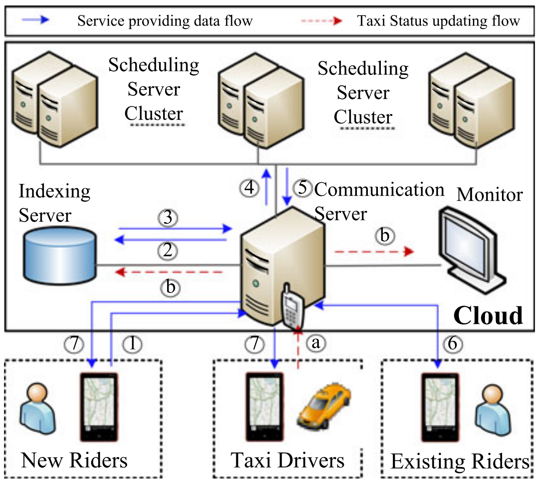

Service providing data flow
Taxi Status updating flow
Scheduling Server Cluster
Scheduling Server Cluster
Indexing Server
Communication Server
Monitor
Cloud
New Riders
Taxi Drivers
Existing Riders
①
②
③
④
⑤
⑥
⑦
ⓐ
ⓑ

Fig. 1. The architecture of the real-time taxi-sharing system.

- *Time window constraints*. All riders that are assigned to V should be able to depart from the origin point and arrive at the destination point during the corresponding pickup and delivery window, respectively.
- *Monetary constraints*. These constraints provide certain monetary incentives for both taxi drivers and riders. That is, a rider does not pay more than without taxi-sharing; a taxi driver does not earn less than without taxi-sharing when travelling the same distance; the fare of existing riders decreases when a new rider joins the trip. We will further discuss the monetary constraints in Section 5.2.

2.3 Objective Function and Problem Definition

Since multiple taxi statues may satisfy a ride request, an objective function is usually applied to find the optimal taxi. A variety of objective functions have been used in the existing literature, where a weighted cost function combining multiple factors such as travel distance increment, travel time increment and passenger waiting time, is the most common [4], [5], [6]. In this study, given a ride request, we aim to find the taxi status which satisfies the ride request with minimum increase in travel distance, formally defined as follows: *given a fixed number of taxis traveling on a road network and a sequence of ride requests in ascending order of their submitted time, we aim to serve each ride request Q in the stream by dispatching the taxi V which satisfies Q with minimum increase in V's scheduled travel distance on the road network.*

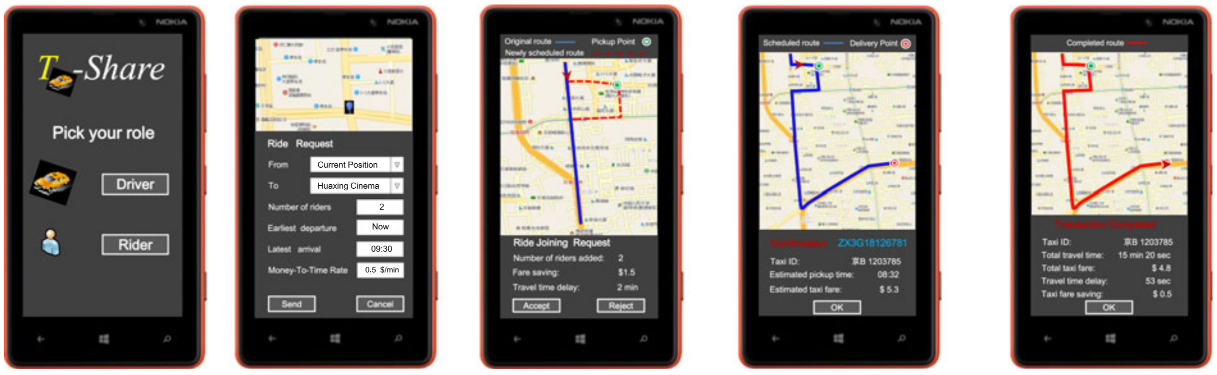
This is obviously a greedy strategy and it does not guarantee that the total travel distance of all taxis for all ride requests is minimized. However, we still opt for this definition due to two major reasons. First, the real-time taxi-sharing problem inherently resembles a greedy problem. In practice, taxi riders usually expect that their requests can be served shortly after the submission. Given the rigid real-time context, the taxi-sharing system only has information of currently available ride requests and thus can hardly make optimized schedules based on a global scope, i.e., over a long time span. Second, the problem of minimizing the total travel distance of all taxis for the complete ride request stream is NP-complete. Please see [3] for a proof.

3 System Architecture

The architecture of our system is presented in Fig. 1. The cloud consists of multiple servers for different purposes and a monitor for administers to oversee the running of the system. Taxi drivers and riders use the same smart phone App to interact with the system, but are provided with different user interfaces by choosing different roles, as shown in Fig. 2a.

As shown by the red broken arrow (a), a taxi automatically reports its location to the cloud via the mobile App when (i) the taxi establishes the connection with the system, or (ii) a rider gets on and off a taxi, or (iii) at a frequency (e.g., every 20 seconds) while a taxi is connected to the system. We partition a city into disjoint cells and maintain a dynamic spatio-temporal index between taxis and cells in the indexing server (detailed in Section 4.1), depicted as the broken arrow (b).

Denoted by the solid blue arrow ①, a rider submits a new ride request Q to the *Communication Server*. Fig. 2b shows the corresponding interface on a rider's smart phone where the blue pin stands for the current location of the rider. All incoming ride requests of the system are streamed into a queue and then processed according to the first-come-first-serve principle. For each ride request Q, the communication server sends it to the *Indexing Server* to search for candidate taxis S_V that are likely to satisfy Q, depicted as the blue arrow ②. Using the maintained spatio-temporal index, the indexing server returns S_V to the communication server, denoted by the blue arrow ③.



Ride Request
From: Current Position
To: Huaxing Cinema
Number of riders: 2
Earliest departure: Now
Latest arrival: 09:30
Money-To-Time Rate: 0.5 $/min

Ride Joining Request
Number of riders added: 2
Fare saving: $1.5
Travel time delay: 2 min

Confirmation ZX3G18126781
Taxi ID: 京B 1203785
Estimated pickup time: 08:32
Estimated taxi fare: $ 5.3

Transaction Completed
Taxi ID: 京B 1203785
Total travel time: 15 min 20 sec
Total taxi fare: $ 4.8
Travel time delay: 53 sec
Taxi fare saving: $ 0.5

(a) Pick a role (b) A new request (c) Ride request notification (d) Ride request confirmation (e) Ride completed

Fig. 2. Screenshots of the mobile client for riders.

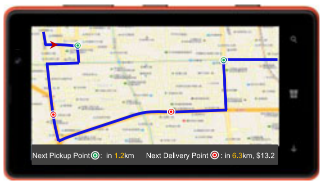

Next Pickup Point: in 1.2km
Next Delivery Point: in 6.3km, $13.2

Fig. 3. Screenshot of the mobile client for drivers.

Represented by the blue arrow ④, the communication server sends ride request Q and the received candidate taxi set S_V to the *Scheduling Server Cluster*. The scheduling cluster checks whether each taxi in S_V can satisfy Q in parallel (detailed in Section 5), and returns the qualified taxi V that results in minimum increase in travel distance and a detailed schedule, shown as arrow ⑤.

Note that in our current implementation, a single machine (see Section 6.2.2 for the specs of the machine) instead of a cluster is used to implement the indexing server and the scheduling servers. The single-machine implementation is able to answer a query in a few milliseconds, i.e., millions of queries per hour. But we believe in that the cluster-based implementation (e.g., using Windows Azure A2 virtual machines) can provide a more robust service. For example, it prevents system crash in case of unexpected power failure of a single machine.

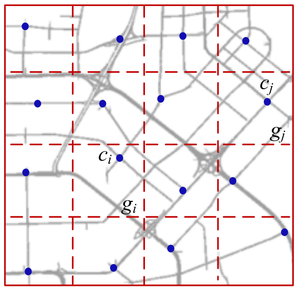
Each rider R who has been already assigned to the taxi V will be enquired whether they would like to accept the join of the new rider, as depicted by blue arrow ⑥. The information, such as the estimated fare saving and travel time increase due to Q's join, will be displayed on their smartphones shown in Fig. 2c. Rider R accepts the route change if she thinks the fare saving is worth the travel time increase. Otherwise, she can veto the route change by clicking the “Reject” button. The system remembers the rider's choice, and automatically rejecting a route change in future if the ratio of the fare saving to the travel time delay is smaller than the largest value the rider has ever rejected. Thus, a taxi passenger will not be bothered often.

After all riders R's who have been already assigned to taxi V accept the route change, the new rider of Q gets a confirmation on her smart phone, as illustrated in Fig. 2d. The confirmation informs the new rider the taxi ID, estimated pickup time and fare, the scheduled route, and a unique reservation code. The new schedule and the same reservation code are sent to the driver's phone at the same time. The reservation code will be used to build a connection between the phones of the new rider and the taxi driver when the new rider gets on the taxi. On the driver side, the smart phone displays a taxi's schedule, e.g., the next pickup and destination points as well as the route, as illustrated in Fig. 3.

(a) ride request tab

(b) taxi tab

Fig. 4. Screenshot of the monitor.



(a) Grid-partitioned map

$$\boldsymbol{M} = \begin{array}{c|ccccc|} & g_0 & g_1 & \cdots\; g_j & \cdots & g_n \\ \hline g_0 & 0 & D_{01} & \cdots\; D_{0j} & \cdots & D_{0n} \\ g_1 & D_{10} & 0 & \cdots\; D_{1j} & \cdots & D_{1n} \\ \vdots & \vdots & \vdots & \vdots & & \vdots \\ g_i & D_{i0} & D_{i1} & \cdots\; D_{ij} & \cdots & D_{in} \\ \vdots & \vdots & \vdots & \vdots & & \vdots \\ g_n & D_{n0} & D_{n1} & \cdots\; D_{nj} & \cdots & 0 \end{array}$$

$$D_{ij} = (t_{ij}, d_{ij})$$

(b) Grid distance matrix

Fig. 5. Grid partitioned map and grid distance matrix.

When a rider's trip is completed, the rider's App will show the exact information, such as the actual fare and travel time, as illustrated in Fig. 2e. The reservation code will be used again to confirm the payment to the Cloud as a transaction ID.

The system administrator oversees the taxi-sharing system via the monitor. The monitor provides two views: one for ride requests, the other for taxis. Fig. 4a shows a screenshot of the ride request view, where all requests are displayed on the map at their corresponding pickup point, with scheduled requests in red and unscheduled requests in blue. On the right, two boxes list the detail information of scheduled and unscheduled ride requests respectively. The search box allows the administrator to quickly locate a request on the map via a request ID.

Fig. 4b shows a screenshot of the taxi view of the monitor. Each taxi is represented by a yellow taxi symbol on the map. The locations of these symbols on the map are updated while the corresponding taxis upload new statuses. Similarly, the search box is used to quickly locate and track a taxi via querying a specific taxi ID.

4 Taxi Searching

The taxi searching module quickly selects a small set of candidate taxis with the help of the spatio-temporal index. In this section, we will first describe the index structure and then detail the searching algorithm.

4.1 Index of Taxis

The spatio-temporal index of taxis is built for speeding up the taxi searching process. Specifically, we partition the road network using a grid. (Other spatial indices such as R tree can be applied as well, but we envision that the high dynamics of taxis will cause prohibitive cost for maintaining such an index.) As shown in Fig. 5a, within each grid cell, we choose the road network node which is closest to the geographical centre of the grid cell as the anchor node of the cell (represented by a blue dot in Fig. 5a. The anchor node of a grid cell g_i is thereafter denoted by c_i. We compute the distance, denoted by d_{ij}, and travel time, denoted by t_{ij}, of the fastest path on the road network for each anchor node pair c_i and c_j. Both the distance and travel time is only computed once. (Alternatively, travel time can be updated dynamically, i.e. calculated once in a while (e.g., every 10 minutes) by leverageing historical data

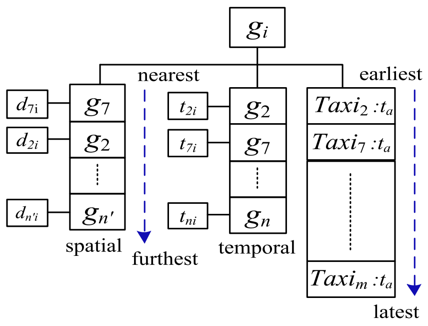


Fig. 6. Spatio-temporal index of taxis.

archives, and travel time estimation techniques e.g., T-Drive [7], [8], [32]. However since the effectiveness of travel time estimation is not a focus of this work, we do not discuss it in details here.) Intuitively, we can use the computed travel time to quickly filter out a large number of taxis whose schedule is "far away" from a given ride request. The distance and travel time results are saved in a matrix as shown in Fig. 5b. The matrix is thereafter referred to as the *grid distance matrix*.

The distance between any two arbitrary nodes is approximated by the distance between two corresponding anchor nodes. In other words, the grid distance matrix provides an approximation of the distance between any two nodes of the road network. These approximated distances avoid the expensive computation cost of frequent fastest path calculations at the stage of taxi searching.

As illustrated in Fig. 6, each grid cell g_i maintains three lists: a *temporally-ordered grid cell list* ($g_i \,.\, l_c^t$), a *spatially-order grid cell list* ($g_i \,.\, l_c^s$), and a *taxi list* ($g_i \,.\, l_v$) . $g_i \,.\, l_c^t$ is a list of other grid cells sorted in ascending order of the travel time from these grid cells to g_i (temporal closeness). Likewise, $g_i \,.\, l_c^s$ is a list of other grid cells sorted in ascending order of the travel distance from these grid cells to g_i (spatial closeness). The spatial and temporal closeness between each pair of grid cells are measured by the values saved in the grid distance matrix shown in Fig. 5b. For example, t_{2i} measures the temporal closeness from g_2 to g_i, and d_{2i} measures the spatial closeness from g_2 to g_i. The spatial grid cell list is only computed once. The temporal grid cell list is computed each time when travel times t_{ij}'s are updated. It is worth mentioning that cells that are neighbours in the grid may not be the neighbours in a grid cell list because the distance is measured in the road network instead of a free space.

The taxi list $g_i \,.\, l_v$ of grid cell g_i records the IDs of all taxis which are scheduled to enter g_i in near future (typically within a temporal scope of 1 or 2 hours). Each taxi ID is also tagged

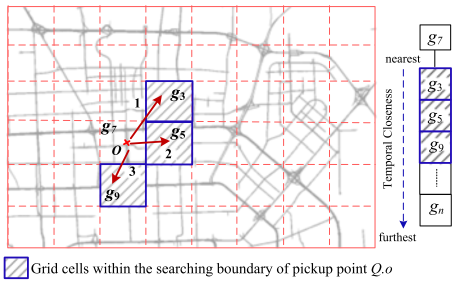

g_7
nearest
1
g_3
g_5
g_9
o
2
3
Temporal Closeness
g_n
furthest
▨ Grid cells within the searching boundary of pickup point $Q.o$

Fig. 7. The single-side taxi searching algorithm.

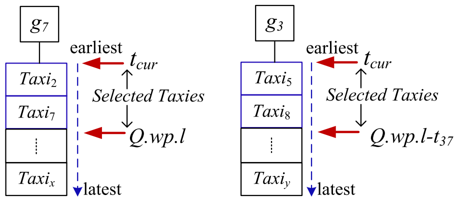


Fig. 8. Choose taxis from the selected grid cells.

with a timestamp t_a indicating when the taxi will enter the grid cell. All taxis in the taxi list are sorted in ascending order of the associated timestamp $t_a \,.\, g_i.l_v$ is updated dynamically. Specifically, taxi V_j is removed from the list when V_j leaves g_i; taxi V_k is inserted into the list when V_k is newly scheduled to enter g_i. If taxis are tracked (see [9]), when new GPS records are received from taxis, taxi lists need to be updated. Specifically, when a new GPS record from V_m is received, denote by g_n the current cell in which V_m is located, the timestamp associated with V_m in the taxi list of cell g_n and cells to be passed by V_m after g_n need to be updated.

4.2 Searching Algorithms

4.2.1 Single-Side Taxi Searching

Now we are ready to describe our first taxi searching algorithm. For the sake of the clarity of description, please consider the example shown in Fig. 7. Suppose there is a query Q and the current time is $t_{cur} \,.\, g_7$ is the grid cell in which $Q \,.\, o$ is located. g_7's temporally-ordered grid cell list $g_7 \,.\, l_g^t$ is shown on the right of Fig. 7. g_7 is the first grid cell selected by the algorithm. Any other arbitrary grid cell g_i is selected by the searching algorithm if and only if Eq. (1) holds, where t_{i7} represents the travel time from grid cell g_i to grid cell g_7. Eq. (1) indicates that any taxi currently within grid cell g_i can enter g_7 before the late bound of the pickup window using the travel time between the two grid cells (if we assume that each grid cell collapses to its anchor node)

$$t_{i7} + t_{cur} \leq Q \,.\, pw \,.\, l \tag{1}$$

To quickly find all grid cells that hold Eq. (1), the single-side searching algorithm simply tests all grid cells in the order-preserved list $g_7 \,.\, l_c^t$ and finds the first grid cell g_f which fails to hold Eq. (1). Then all taxis in grid cells before g_f in list $g_7 \,.\, l_c^t$ are selected as candidate taxis.

In Fig. 7, grid cell g_3, g_5 and g_9 are selected by the searching algorithm. Then for each selected grid cell g_s, the algorithm selects taxis (in $g_s \,.\, l_v$) whose t_a is no later than $Q \,.\, wp.l - t_{s7}$. For instance, Fig. 8 shows how taxis are selected from grid cell g_7 and g_3.

The taxi which can satisfy Q with the smallest increase in travel distance must be included in one of the selected grid cells (under the assumption that each grid cell collapses). Unfortunately, this algorithm only considers taxis currently "near" the pickup point of a query (thus called single-side search). As the number of selected grid cells could be large, this algorithm may result in many taxis retrieved for the later scheduling module (therefore increasing the entire computation cost), which is certainly not desirable for a rigid real-time application like taxi ridesharing. Actually, the

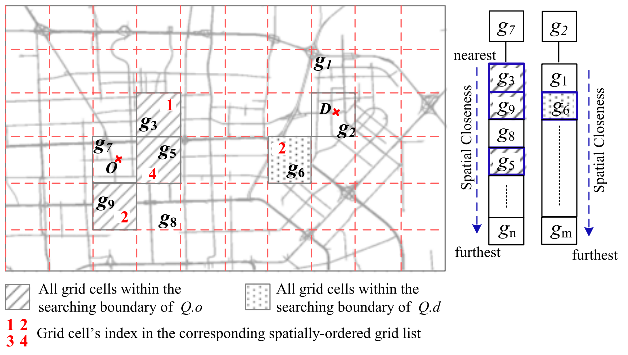

g1
g2
g3
g5
g6
g7
g8
g9
O
D
nearest
furthest
Spatial Closeness
All grid cells within the searching boundary of Q.o
All grid cells within the searching boundary of Q.d
Grid cell's index in the corresponding spatially-ordered grid list

Fig. 9. Overview of the dual-side taxi searching algorithm.

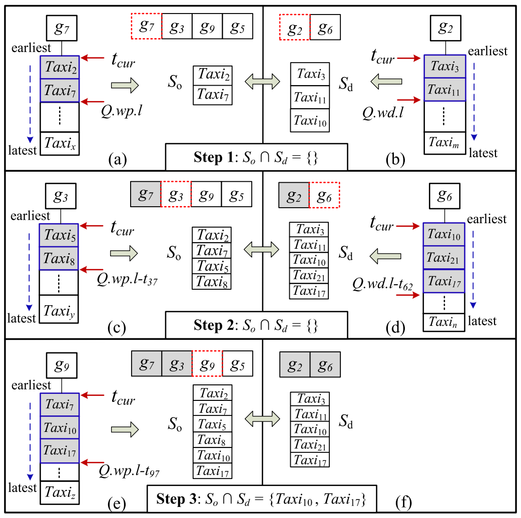


Fig. 10. Calculation of the taxi set in the taxi searching process.

spatiotemporal factor on the destination point of queries also provides us with opportunities to reduce the number of grid cells to be selected. Along this idea, we propose a dual-side searching algorithm as an effort for striking a balance between the distance optimality and the computation cost.

4.2.2 *Dual-Side Taxi Searching*

The dual-side searching is a bi-directional searching process which selects grid cells and taxis from the origin side and the destination side of a query simultaneously.

To dive into the details of the algorithm, consider the ride request illustrated in Fig. 9 where g_7 and g_2 are the grid cells in which $Q.o$ and $Q.d$ are located respectively. Squares filled with stripes stand for all possible cells searched by the algorithm at $Q.o$ side. These cells are determined by scanning $g_7.l_c^t$, the temporally-order grid cell list of g_7. That is, each grid cell in $g_7.l_c^t$ which holds Eq. (2) is a candidate cell to be searched at the origin side. Eq. (2) indicates that any taxi currently within grid cell g_i can enter g_7 before the late bound of the pickup window using the latest travel time between the two grid cells (assuming each grid cell collapses to its anchor node). The red number in each such grid cell indicates its relative position in $g_7.l_c^s$, the spatially-ordered grid list of g_7

$$t_{cur} + t_{i7} \leq Q.dw.l. \tag{2}$$

Squares filled with dots indicate the candidate grid cells to be accessed by the searching algorithm at $Q.d$ side. Likewise, each such grid cell g_j is found by scanning $g_2.l_c^t$ to select all grid cells which holds Eq. (3), which indicates that any taxi currently in g_j can enter the g_2 before the late bound of the delivery window (assuming that each grid cell collapses to its anchor node). In this example, g_6 is the only satisfying grid cell as shown by Fig. 9

$$t_{cur} + t_{j2} \leq Q.dw.l. \tag{3}$$

Fig. 10 then illustrates the searching process step by step. The algorithm maintains a set S_o and a set S_d to store the taxis selected from $Q.o$ side and $Q.d$ side respectively. Initially, both S_o and S_d are empty. The first step in the searching is to add the taxis selected from taxi list $g_7.l_v$ to taxi set S_o as depicted in Fig. 10a, and add the taxis selected from taxi list $g_2.l_v$ to taxi set S_d as depicted by Fig. 10b. Then the algorithm calculates the intersection of S_o and S_d. If the intersection is not empty, the algorithm stops immediately and returns the intersection set. Otherwise, it expands the searching area by including one other grid cell at each side at a time.

To select next cells, we use the following heuristic: for a taxi V, the closer a cell to be passed by V is to g_7 and the closer a cell to be passed by V is to g_2 (measured in the distance between the anchor nodes of the cells), the smaller V's scheduled travel distance increases after the insertion of the ride request. For the purpose of minimizing increased travel distance, the next grid cell included at $Q.o$ side is always chosen as the next element in the spatially-ordered grid list $g_7.l_c^s$ which holds Eq. (2). Similarly, the next grid cell included at $Q.d$ side is always chosen as the next element in the spatially-ordered grid list $g_2.l_c^s$ which holds Eq. (3).

In this example, since S_o and S_d produces an empty intersection, the algorithm expands at $Q.o$ side to include g_3 (indicated by the broken red rectangle) and add taxis selected from $g_3.l_v$ as depicted in Fig. 10c. At $Q.d$ side, the algorithm covers g_6 and adds taxis as indicated in Fig. 10d. Unfortunately, the intersection set of S_o and S_d remains empty. Consequently, the algorithm needs to continue expanding the searching area at both sides. g_9 is then selected at $Q.o$ side; but no grid cell can be further included at the $Q.d$ side. After adding the taxis selected from $g_9.l_v$ into set S_o as shown in Fig. 10e, we find $Taxi_{10}$ and $Taxi_{17}$ as the intersection between S_o and S_d. Hence, the searching algorithm terminates.

Compared to the dual side searching algorithm, the disadvantage of the single side searching algorithm is that the number of selected grid cells could be large and thus it results in many taxis retrieved for the later scheduling process. In other words, it increases the overall computation cost, which is certainly not desirable for a rigid real-time system like taxi-sharing. Though the dual-side searching algorithm may result larger increase in travel distance for the given ride request, as a compensation for the small loss in distance optimality, the algorithm selects far fewer taxis for the schedule allocation step, reducing the computation cost and ride request processing time. We found in the experiments that the number of selected taxis is reduced

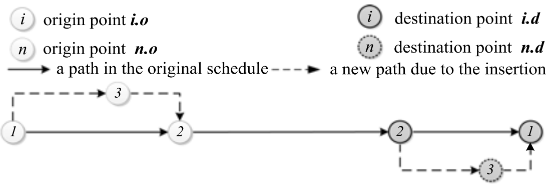


Fig. 11. One possible insertion of a ride request into a schedule.

by 50 percent while the increase in travel distance is just 1 percent over the single-side search algorithm.

5 Taxi Scheduling

Given the set of taxi statuses S_V retrieved for a ride request Q by the taxi searching algorithm, the purpose of the taxi scheduling process is to find the taxi status in S_V which satisfies Q with minimum travel distance increase.

To this end, given a taxi status, theoretically we need to try all possible ways of inserting Q into the schedule of the taxi status in order to choose the insertion which results in minimum increase in travel distance. All possible ways of insertion can be created via three steps: (i) reorder the points in the current schedule, subject to the precedence rule, i.e., any origin point precedes the corresponding destination point (we refer to this step as the schedule reordering thereafter); (ii) insert $Q.o$ into the schedule (iii) insert the $Q.d$ into the schedule. The capacity and time window constraints are checked in all three steps, during which the insertion fails immediately if any constraint is violated. The monetary constraints are then checked for the insertion after all three steps have been done successfully. Finally, among all insertions that satisfy all constraints, we choose the insertion that results in minimum increase in travel distance for the given taxi status.

Consider a schedule with n points, among which m points are pickup points. After the first step, i.e., the schedule reordering step, there will be as many as $n!/2^m$ sequences which comply with the precedence rule. ($n!$ is the total number of squences of all n points. Since each pickup point must precede the correspoinding destination point, so the total number of squences needs to be divided by 2^m.) Though reordering the schedule is theoretically necessary for finding the optimal insertion way, we find that it is not the case in practice via experiments (see Section 6.2.3). Therefore, for the sake of simplicity, we do not consider the schedule reordering step here.

Next we describe how to check the feasibility of each insertion possibility, subject to the capacity and time window constraints first (Section 5.1) and then the monetary constraints (Section 5.2), given a pair of Q and V. A computer cluster can be employed to parallelize the computation by assigning taxi statuses to different computers in the cluster, so the constraints checking for multiple taxi statuses can be performed simutaneously.

5.1 Time Window Constraints

Given a schedule of n points, there is clearly $O(n^2)$ ways to insert a new ride request into the schedule. For example, Fig. 11 shows one way of inserting a request into a schedule with four points. To insert $Q_3.o$ after point $Q_1.o$ optimally, the algorithm needs to find the first path (starting from the shortest path) from $Q_1.o$ to $Q_3.o$ which allows the taxi to arrive at $Q_3.o$ during $Q_3.pw$ given the scheduled arrival time at $Q_1.o$. Since the shortest path is often not the fastestt one when considering real road traffic, it is likely that multiple paths needs to be calculated before finding the first satisfactory path from $Q_1.o$ to $Q_3.o$. Similar process is required for other connecting paths, as illustrated by the dash lines in Fig. 11. As a result, the overall computation load can be extremely high for checking just one insertion way. To ease the computation load, here we only consider using the fastest path from one point to another during the insertion, though the new route may not be the shortest one in theory.

Denote by $\rightarrow$ the travel time of the fastest path from one location to another location, and t_w represents the time spent waiting for the passenger if the taxi arrives $Q_3.o$ ahead of $Q_3.pw.e$. Eq. (4) gives the travel time delay, denoted by t_d after inserting $Q_3.o$ between $Q_1.o$ and $Q_2.o$

$$t_d = (Q_1.o \rightarrow Q_3.o) + (Q_3.o \rightarrow Q_2.o) + t_w - (Q_1.o \rightarrow Q_2.o) \tag{4}$$

If t_d results in the late arrival at point $Q_2.o$ or any point after $Q_2.o$ in the original schedule, then the insertion fails. For this purpose, we introduce the notion of slack time. Denote by a_p and a_d the projected arrival time at a pickup point $Q.o$ and a destination point $Q.d$, respectively. Then the slack time at $Q.o$ and $Q.d$, denoted by $(Q.o)_{st}$ and $(Q.d)_{st}$ respectively, is calculated by Eq. (5) and Eq. (6), respectively

$$(Q.o)_{st} = Q.pw.l - a_p \tag{5}$$

$$(Q.d)_{st} = Q.dw.l - a_d \tag{6}$$

Thus, we can use slack times as a shortcut to check whether the delay incurred due to an insertion destroys the timely arrivals at any subsequent point in the schedule. In the example shown by Fig. 11, if $t_d \geq Min\{(Q_1.d)_{st}, (Q_2.d)_{st}\}$, then the insertion fails. If $Q_3.o$ is inserted successfully, the system proceeds to insert $Q_3.d$ in a similar way. Algorithm 1 summaries the process of computing a new route for a possible insertion way, represented by a pair (i, j). This algorithm is run for all possible insertion ways.

```
Algorithm 1: Computing the new schedule and route after an insertion
Data: Ride request $Q$, taxi status $V$, insertion position $i$ for $Q.o$,
      insertion position $j$ for $Q.d$, current time $t_{cur}$
Result: Return new_schedule if the insertion succeeds; otherwise return
        False.
1  if $t_{cur} + (V.l \rightarrow Q.o) > Q.pw.l$ then /* cannot arrive $Q.o$ on time */
2      return False
3  if the time delay incurred by the insertion of Q.o causes the slack time of
   any point after position i in schedule s smaller than 0 then
4      return False
5  new_schedule ⟵ insert $Q.o$ into $V.s$ at position $i$   /* the slack time
   of each pickup(delivery) point after position $i$ is also
   updated accordingly */
6  $t_j$ ⟵ the scheduled arrival time of the $j^{th}$ point of $V.s$
7  $l_j$ ⟵ the geographical location of the $j^{th}$ point of $V.s$
8  if $t_j + (l_j \rightarrow Q.d) > Q.dw.l$ then   /* cannot arrive $Q.d$ on time */
9      return False
10 if the time delay incurred by the insertion of Q.d causes the slack time of
   any point after position j in new_schedule smaller than 0 then
11     return False
12 new_schedule ⟵ insert $Q.d$ into new_schedule at position $j$     /* the
   slack time of each pickup(delivery) point in new_schedule is
   also updated accordingly */
13 return new_schedule
```

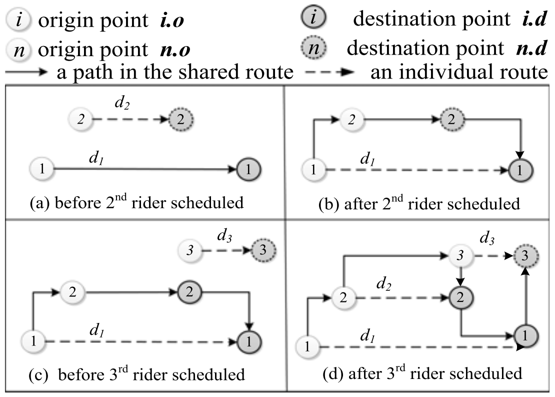

origin point i.o
destination point i.d
origin point n.o
destination point n.d
a path in the shared route
an individual route
(a) before 2nd rider scheduled
(b) after 2nd rider scheduled
(c) before 3rd rider scheduled
(d) after 3rd rider scheduled

Fig. 12. An example of the pricing constraint.

5.2 Monetary Constraints

The new schedule after the insertion, so far, has only been checked against the capacity and time window constraints. It should also meet the monetary constraints. In this section we formulate the monetary constraints of taxi-sharing.

On one hand, we impose two constraints which encourage riders to participate in taxi-sharing by rewarding them with certain monetary gains. The first rider monetary constraint says that any rider who participates in taxi-sharing should pay no more than what she would pay if she takes a taxi by herself. The second rider monetary constraint says that if an occupied taxi V is to pick up a new rider Q, then each rider P currently sitting in V whose travel time is lengthened due to the pickup of Q, should get a decrease in taxi fare; and the fare decrease should be proportional to P's increase in travel time.

On the other hand, we enforce one constraint which gives the driver motivation to participate in taxi-sharing. This constraint says that a driver should charge for all distances she has travelled. Intuitively the driver should make money for the distance of reroutes incurred by the join of any new passenger.

Now let us consider these three monetary constraints together in the scheduling context: given a taxi status V and a new ride request Q_n, under what conditions will V satisfy the above three monetary constraints with respect to Q_n.

Denote by $Q_1, \ldots Q_{n-1}$ the riders involved in the current schedule of V before the join of Q_n. Also denote by d_i the distance between $Q_i.o$ and $Q_i.d$ on the road network $, i = 1, \ldots, n$. Denote by f_i the taxi fare of rider Q_i if V picks up Q_n. Denote by $F : R^+ \to R^+$ the fare calculation function, which maps the travelled distance to the taxi fare. Function F can be defined by some transportation authority or taxi company. Then the first monetary constraint can be expressed by Eq. (7)

$$f_i \leq F(d_i), i = 1, \ldots, n. \tag{7}$$

Denote by M the revenue of the driver if she picks up Q_n and by D the travel distance of the new route after the pickup. Then the driver monetary constraint is expressed by Eq. (8)

$$M \geq F(D). \tag{8}$$

Since $M = \Sigma f_i$, we then have Eq. (9) by bridging the two equations above

$$F(D) \leq M = \Sigma f_i \leq \Sigma F(d_i), i = 1, \ldots, n. \tag{9}$$

M can take any value between $F(D)$ and $\Sigma F(d_i)$ in order to make Eq. (9) hold. In this paper, we take the lower bound $F(D)$ in order to minimize the total taxi fare of riders. Therefore, we have $M = F(D)$.

Then we need to distribute the total fare M to each individual rider. Denote by Δf_i the decrease in taxi fare for rider Q_i, and ΔT_i the increase in travel time of rider Q_i due to the pickup of Q_n, $i = 1, \ldots, n-1$. The fare is determined in the way expressed by Eqs. (10) and (11), where f_n is the taxi fare of rider Q_n, ΔD is the travel distance increase of the taxi route due to the pickup of Q_n and $f \geq 0$ is some constant

$$f_n = F(d_n) - f, \tag{10}$$

$$\Delta f_i = \frac{\Delta T_i}{\sum_{i=1}^{n-1} T_i} [(F(d_n) - f) - F(\Delta D)], i = 1, \ldots, n-1. \tag{11}$$

Eq. (10) says that the new rider pays by f less than whatever she would pay if she rides alone. Eq. (11) says that (i) existing riders collectively save an amount which equals the difference between the charge of the new rider and the driver's expected fare increase due to the increase in travel distance; and (ii) existing riders split the total saving proportional to their individual travel time delay (the second rider monetary constraint). Since it requires $\Delta f_i \geq 0$, therefore, we have Eq. (12)

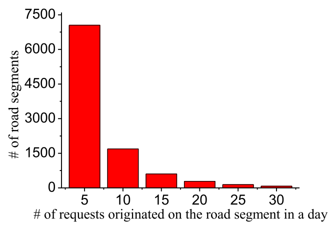
$$F(d_n) \geq F(\Delta D) + f. \tag{12}$$

Eq. (12) by itself is the sufficient and necessary condition for taxi V to satisfy all three monetary constraints with respect to Q_n.

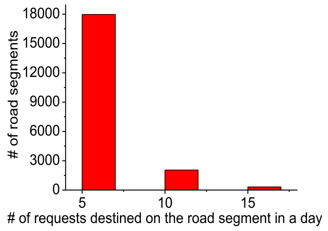
Fig. 12 illustrates how to apply the monetary constraints with a concrete example. Fig. 12a shows the schedule of a taxi before the second rider boards. The fare of the first rider is $f_1 = F(d_1)$. The monetary constraint for picking up the second rider is $F(d_2) \geq F(\Delta D) + f$, where ΔD is the increase in travel distance due to the join of the second rider. If the above constraint stands, then we have $\Delta f_1 = [F(d_2) - f] - F(\Delta D)$ and $f_2 = F(d_2) - f$. Likewise, Fig. 12c shows the schedule of the taxi after the second rider joins and before the third rider joins. Similarly, the pricing constraint for picking up the third rider is $F(d_3) \geq F(\Delta D') + f$, where $\Delta D'$ is the increase in travel distance due to the join of the third rider. If this constraint stands, then we have $\Delta f_i = \frac{\Delta T_i}{\Delta T_1 + \Delta T_2}[F(d_3) - f] - F(\Delta D')), i = 1, 2$ and $f_3 = F(d_3) - f$.

Some riders may think the taxi fare decrease is not worth the increase in travel time and thus rejects the pickup decision. We thus introduce a parameter $Q_i.r$ for each rider Q_i, which presents Q_i's acceptable money-to-time rate. That is to say, Q_i supports the pickup of a new rider only when the ratio of the fare decrease to the travel time increase is larger than $Q_i.r$. The above constraint is expressed by Eq. (13). And an insertion satisfies the monetary constraints only when all current riders on the taxi support the pickup decision

$$\Delta f_i / \Delta T_i \geq Q_i\,.\,r. \tag{13}$$

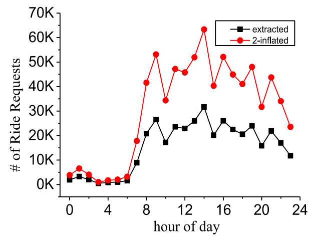


(a) Origins

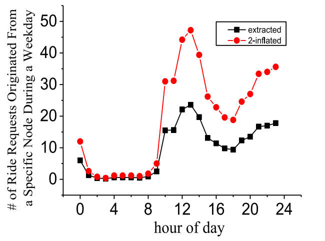


(b) Destinations

Fig. 13. Distribution of ride requests over road segments.



(a) from all road segments



(b) from a particular road segment

Fig. 15. Inflated and extracted number of ride requests during a day.

6. Evaluation

6.1 Setting

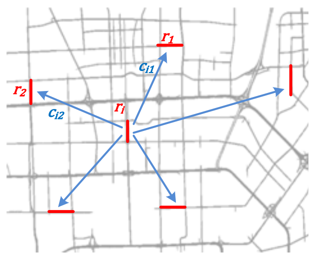
6.1.1 Data Set

Road networks: We perform the experiments using the real road network of Beijing, which contains 106,579 road nodes and 141,380 road segments.

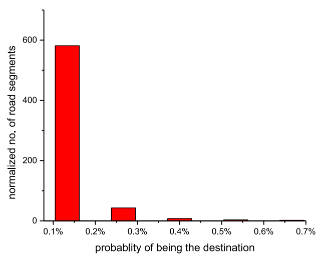
Taxi Trajectories: The taxi trajectory data set contains the GPS trajectory of over 33,000 taxis during a period of 87 days spanning from March to May in the year of 2011. The total distance of the data set is more than 400 million kilometres and the number of points reaches 790 million. After trip segmentation, there are in total 20 million trips, among which 46 percent are occupied trips and 54 percent are non-occupied trips. We map each occupied trip to the road network of Beijing using the map-matching algorithm proposed in [10]. Fig. 13 shows the distribution of pickup and destination points of the ride requests in the data set over road segments in a day (note that long tails, i.e., road segments with large number of requets, are not shown in the figures due to space limitation). It is clear that ride requests are distributed sparsely over the road network.

6.1.2 Experimental Platform

In order to validate our proposed system under practical settings, instead of generating random ride requests and initial taxi statuses, we mine the trajectory data set to build an experimental platform. From the historical trajectory data set, the platform learns information regarding 1) the distribution of the ride requests on the road network over time of day, and 2) the mobility patterns of the taxis. With this learned knowledge, the platform then generates a realistic ride request stream (meaning that the origin-destination pairs and time windows of ride requests follow the learned distribution) and initial taxi statuses for our experiments. We envision that this platform (available at http://cs.uic.edu/~sma/ridesharing/) can be applied to many other urban and transport computation problems.



(a) Different destination road segments for requests originated from r_i



(b) Distribution of destination road segments for requests originiated from a specific road segment

Fig. 14. Transition probability of road segment r_i during a time frame f_j.

Ride request stream. The goal is to generate real-time ride requests that are as realistic as possible. For this purpose, we first discretise one day into small time frames, denoted by f_j's. Denote all road segments by r_i's. We assign all historical ride requests into time frames. Assume that the arrivals of ride requests on each road segment approximately follow a Poisson distribution during time frame f_j. Thus, we can learn λ_i^j, i.e., the parameter of the Poisson distribution for road segment r_i during time frame f_j. Specifically, for each road segment r_i, we count the number of ride requests that originate from r_i within time frame f_j, denoted by c_i^j. Then we calculate λ_i^j based on c_i^j using Eq. (14) (where $len(f)$ is the length of a frame in time units) and generate a ride request stream that follows a Poisson process with parameter λ_i^j. In order to generate destination of requests trufhfully, for each c_i^j, we decompose it into an array of numbers $\{c_{i1}^j, c_{i2}^j, \dots c_{im}^j\}$, where $c_{ik}^j, k = 1, 2, ..m$ represents the number of requests which are originated from road segment r_i and destined for road segment r_k during time frame f_j, as illustrated by Fig. 14a. Therefore, the transition probability from r_i to r_k during time frame f_j, denoted by p_{ik}^j, can be estimated using Eq. (15). Fig. 14b shows the distribution of destination road segments for requests that originate from road segment r_i in an hour

$$\lambda_i^j = c_i^j / len(f), \tag{14}$$

$$p_{ik}^j = c_{ik}^j / c_i^j. \tag{15}$$

For each ride request Q generated in frame f_j with the origin road segment being r_i, the destination road segment is generated according to the transition distribution p_{ik}^j. $Q.pw.e$ and $Q.dw.e$ equals to $Q.t$, i.e., the submitted time of the ride request. $Q.pw.l$ is calculated by applying a fixed window size. $Q.dw.l$ equals to the sum of $Q.pw.l$ and the average travel time between the origin and destination pair learned from the GPS trajectory data set.

Note that the taxi GPS trajectory data set only reveals the number of ride requests that got served. In reality there are also many ride requests unsatisfied and disappeared due to the shortage of taxis. To take such ride requests into consideration, we introduce a system parameter Δ, supposing that the number of real ride requests is Δ times the number of request extracted from the trajectory data set. Figs. 15a and 15b show the supposed number and the extracted number of ride requests that originate from all

TABLE 1
Parameter Setting for Ride Request Generation

Notation	Definition	Value
t_s	The start time of simulation	09:00
t_e	The end time of simulation	09:30
$\#taxi$	The number of taxis	7088
$\#taxi_o$	The number of taxis occupied initially	4722
ws	The window size	5 min
$len(f_i)$	The length of a time frame	60 min

road segments and a specific hot spot road segment (i.e., with a large number of requests), respectively, over time of a "hot day" (i.e. with more requests than an average day), where the time frame is 1 hour and $\Delta = 2$.

Initial taxi statuses. To keep the characteristics of the realistic scenario, we use the real taxi statuses by slicing the historical trajectories at a certain timestamp. Specifically, we select a date and choose a particular second of day as the timestamp when the experiment starts, denote it by t_s. We scan all the GPS records of the selected date to determine the initial states of taxis. A taxi status V is set to be occupied if it is recorded occupied crossing timestampt_s. The initial schedule of V can be initialized according to the record. A taxi V is set to be vacant if it is recorded vacant both just before and right after t_s. The concept of "just before" and "right after" is controlled by a temporal parameter, which is set to be 10 minute. All remaining taxis are then considered as not recorded and thus not used in the simulation.

The ride requests and initial states used in all experiments are generated with parameters listed in Table 1.

6.1.3 Framework

We study two strategies in the searching algorithm (single-side and dual-side) and two strategies in the scheduling algorithm (first-fit and best-fit), resulting in four taxi-sharing methods. We compare the performance of these four methods with that of a non-taxi-sharing method as the number of requests (i.e., Δ) changes. We also test the performance of these four methods by changing the money-to-time rate parameter of the monetary constraints, and study the necessity of the schedule reordering step (i.e., considering different pickup and drop-off orders) in the scheduling algorithm.

6.1.4 Baseline Methods

The *Non-Taxi-sharing method (NR)* forbids taxi-sharing and assumes that a vacant taxi moves to pick up the rider that it can pick up at the earliest time.

Taxi searching step. A taxi-sharing method is single-side if the taxi searching algorithm retrieves taxis only from the origin side of a request; otherwise, it is dual-side.

Taxi scheduling step. A taxi-sharing method is called best-fit where the taxi scheduling process tries all candidate taxis returned by the taxi searching algorithm. Otherwise, is called first-fit if the scheduling process terminates once it finds a taxi that satisfies the ride request.

Because the two choices can be made independently, we get the following four taxi-sharing methods: *Single-side and First Fit Taxi-sharing (SF), Single-side and Best-fit Taxi-sharing (SB), Dual-side and First Fit Taxi-sharing (DF), Dual-side and Best-fit Taxi-sharing (DB).*

The fare calculation function $F = pD$, where D is the traveled distance and p is some constant price for a unit traveled distance. The money-to-time rates of ride requests are assumed to follow an exponential distribution with a mean value m.

6.1.5 Measurements

The performance of the taxi-sharing system is evaluated in two perspectives, namely effectiveness and efficiency. We first describe following effectiveness measurements.

Relative distance rate (RDR). Define the distance of a ride request Q as the distance between its origin point $Q.o$ and its destination point $Q.d$. Denote by D_{SR} the sum of distances of ride requests that get satisfied and by D_V the total distance travelled by all taxis while being occupied in a taxi-sharing method. RDR is calculated by Eq. (16)

$$RDR = D_V / D_{SR}. \tag{16}$$

RDR evaluates the effectiveness of taxi-sharing by measuring how much distance is saved compared to the case where no taxi-sharing is used. The value of *RDR* can be any positive number. The smaller *RDR* is, the more vehice distance the ridesharing system saves. When the value is greater than 1, it indicates that the ridesharing system does not save but increases the total travel distance.

Satisfaction rate (SR). Is the ratio of the number of ride requests that get satisfied to the total number of ride request (exclude ride requests that are already served by taxis at the initial state in the ride request counting). Thus, $SR \in [0, 1]$. The larger *SR* is, the more requests the ridesharing system has satisfied.

Seat occupancy rate (SOR). Measures the seat occupancy rate in all taxis during a given time period. Denote by C the number of passenger seats in a taxi, by N the number of taxis, by T a period of time, and by T_s the sum of the travel time of all requests that are satisfied during period T. *SOR* is calculated by Eq. (17). *SOR* provides an inituitive perception of how well the seats in the vehicles are utilized. *SOR* ranges from zero to one. When *SOR* equals to 1, it means a fully utilizated outcome where each seat in every vehicle is occupied all the time

$$SOR = T_S / (N \times C \times T). \tag{17}$$

Taxi-sharing rate (TR). Is the percentage of ride requests participating in taxi-sharing among all satisfied requests. The value of *TR* ranges from 0 to 1. A larger value indicates more ridesharing opportunities created.

Fare saving rate (FSR). Is the average saving percentage in taxi fare of riders who participate in taxi-sharing. The value may range from 0 to 1 with 1 meaning that all ridesharing riders save the whole fare and get free rides.

Now we introduce following efficiency measurements.

Number of road nodes accessed per ride request (#RNAPR). Is the number of accessed road network nodes per ride request (due to the shortest path calculations).

Number of grid cells accessed per ride request (#GCAPR). Is the number of accessed grid cells per ride request.

TABLE 2
Default Values of Parameters Used in Experiments

Definition	Value
Beijing Map Size	32 *40 km^2
The size of grid (i.e., number of grid cells)	30*30
Schedule reordering before insertion	no
p, taxi fare per kilometer	¥2
C, no. of passenger seats in a taxi	3
m, mean of the money-to-time rate distribution	¥0/min

Num. of taxis accessed per ride request (#TAPR). Is the no. of taxis accessed by the scheduling module per request. *RNAPR, GCAPR, TAPR* are machine-independent indicators for computation cost of the system.

Execution time per ride request. Is the CPU time spent for serving each ride request. It consists of taxi searching time (elapsed between step ② and ④ in Fig. 1) and taxi scheduling time (elapsed between ⑤ and ⑦ in Fig. 1).

6.2 Experimental Results

Table 2 lists default values for experiment parameters.

6.2.1 Effectiveness

Fig. 16a shows SR, i.e., the satisfaction rate, of all methods as Δ changes. All ridesharing methods first show a slight increase and then keep declining in SR as Δ increases. This is because as the value of parameter Δ starts increasing, the ridesharing opportunies increase as well. When Δ is small, the increase in ridesharing opportunies is larger than the increase in the number of request, as a result, the satisfaction rate surges. When Δ gets larger, ridesharing opportunies do not emerge as fast as new requests arrive, therefore, the satisfiaction rate starts dropping. In contrast, the non-ridesharing method NR can only suffers from the increase in Δ, i.e., resulting in a decreasing SR. It is clear that all flavours of taxi-sharing methods have a considerably higher satisfaction rate (about 23 percent higher on average) than the NR method for all Δ values. The difference in the satisfaction rate among taxi-sharing methods is insignificant as no particular technique is proposed for minimizing the satisfaction rate.

Fig. 16b shows TR, the percentage of ride requests participating in taxi-sharing among all satisfied ride requests, for all taxi-sharing methods. Not surprisingly, TR surges as Δ increases. This is because taxi-sharing opportunities are likely to rise as the number of taxi ride request increases. Consequently, more ride requests can be satisfied via taxi-sharing. But since the total number of ride requests increases even faster, the satisfaction rate still drops, as illustrated by Fig. 16a.

To calculate the fare of riders in the experiments, we instantiate the fare calculation function as follows: the fare charged by a taxi driver is linear to the distance travelled by the taxi, i.e., the product of the distance travelled and fare per unit distance. Fig. 16c shows that FSR, the average fare saving of riders who participate in taxi-sharing, drops as Δ increases. The larger Δ is, the more passengers participate in ridesharing. However this does not necessarily guarantee that the average saving of the participants increases. The average cost of this amount saving is about 5.08 minute delay in travel time. We believe that most riders are willing to tolerate this amount of delay, especially under the high request demand scenarios in which this taxi-sharing system is most likely to be useful.

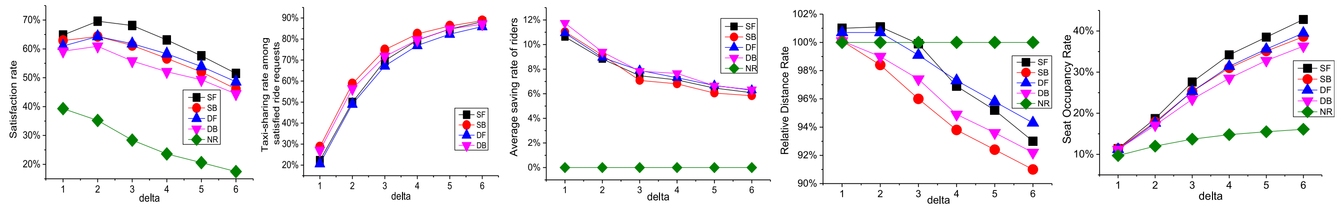
Fig. 16d shows RDR steadily drops as parameter Δ increases. Again, this is likely because as the number of ride requests increases, more ride requests can share partial trips with each other and thus more distance the taxi-sharing methods save. The SB taxi-sharing method outperforms other methods, since SB reduces the increase in travel distance most. The DB taxi-sharing method slightly trails SB method as the taxi searching step of it explores fewer grid cells. Two first-fit based taxi-sharing methods show clearly higher relative distance rate. From the picture, we can see that taxi-sharing methods save up to 12 percent in travel distance, depending on delta. Given the fact that there are 67,000 taxis in Beijing (not in the data set) and each taxi runs 480 km per day (learned from the data set), the saving achieved by taxi-sharing here means over 1.5 billion kilometres in distance per year, which equals to 120 million liter of gas per year (supposing a taxi consumes 8 liter of gasoline per 100 km) and 2.2 million of carbon dioxide emission per year (supposing each liter of gas consumption generates 2.3 kg of carbon dioxide).

Fig. 16e shows SOR increase as parameter increases. This is easy to understand since as the number of requests increases, more ridesharing opportunies appear. As a result, SOR increases as well. The differences between ridesharing methods in SOR are not significant since these methods are not designed for optimizing SOR.

Figs. 17a, 17b, 17c, 17d and 17e shows SR, TR, FSR, RDR and SOR of taxi-sharing methods for different mean values of the money-to-time rate of ride requests, respectively, when $\Delta = 1$. All measurements except FSR show a decrease tendency as the mean money-to-time rate increases. When the mean money-to-time rate increases from ¥0.25/min to ¥0.5/min, the decrease is the largest.

6.2.2 Efficiency

Tested on a single server with 2.67 GHz CPU and 16 GB RAM (using a single thread), the average taxi searching



(a) *SR* (b) *TR* (c) *FSR* (d) *RDR* (e) *SOR*

Fig. 16. Performance in effectiveness measurements of different methods.

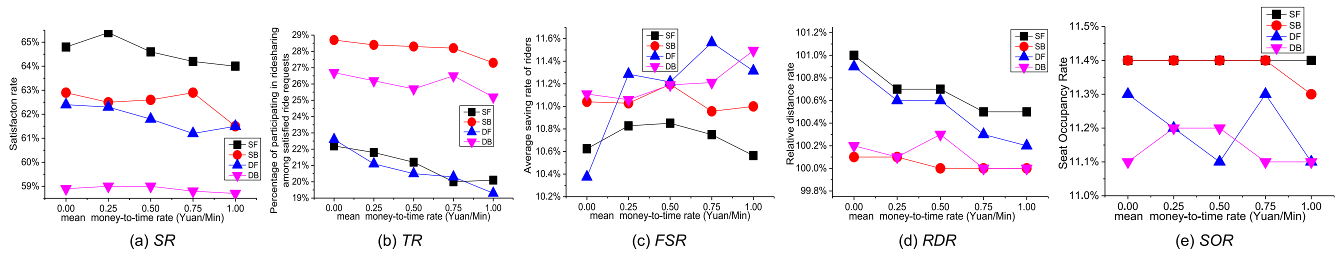


Fig. 17. Performance in effective measurements versus money-to-time rate.

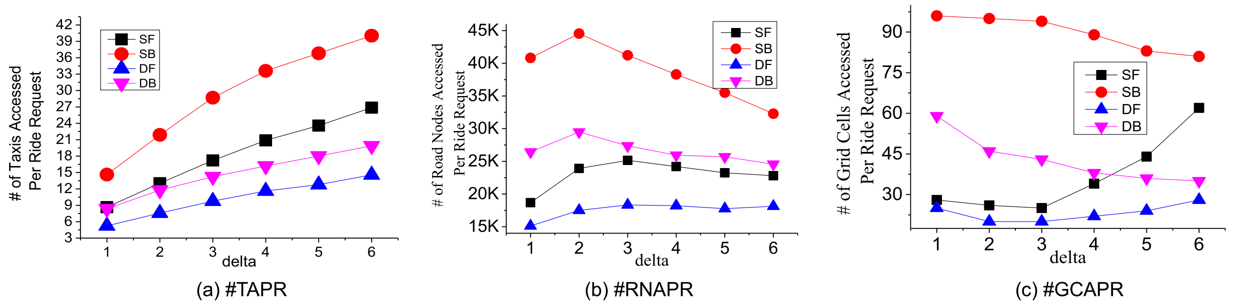


Fig. 18. Computation cost in terms of node access per ride request.

time and scheduling time for the *DB* method is 0.15 and 10.33 ms, respectively.

We also test the efficiency of the system using machine-independent measurements. The three sub-graphs of Fig. 18 show the number of taxis accessed per ride request, the number of road nodes accessed per ride request, and the number of grid cells accessed per ride request, respectively, for different taxi-sharing methods under different Δ. It is clear from the pictures that all taxi-sharing methods do not show sharp increase in computation cost as Δ increases. It is also obvious that the computation cost of the *DB* taxi-sharing method is significantly smaller than that of *SB* taxi-sharing method. Especially when $\Delta \geq 4$, the computation cost of the *DB* method is even smaller than that of the *SF* method. The result of Figs. 16 and 18 together validate our motivation for the dual-side taxi searching algorithm. That is, the dual-side searching indeed incurs small increase in travel distance in exchange for the significant decrease in computation cost.

6.2.3 Necessity of the Schedule Reordering

Fig. 19 shows the average execution time per ride request under different values of Δ when using the *DB* taxi-sharing method with and without the schedule reordering before insertion. The execution time per ride request is about 20 percent longer on average when the schedule reordering is performed.

Meanwhile there is almost no change in all effectiveness measurements. From the results, we also learned that in practice it is extremely rare that the optimal insertion requires the schedule reordering. Although the execution time per ride request remains a reasonable small value with the schedule reordering step, there is still no incentive to do so in practice.

7 Related Work

We study three categories of related works, positioning our work in the research community.

7.1 Taxi Recommendation and Dispatching

Quite a few recommender systems have been proposed for improving an individual taxi driver's income and reducing unnecessary cruising. Based on historical taxi trajectories, Yuan et al. [11] proposed a system that suggests some parking places for an individual taxi driver towards which they can find passengers quickly and maximize the profit of the next trip. Similarly, Ge et al. [12] suggests a sequence of pickup points for a taxi driver. While these systems are only designed from the perspective of taxi drivers, our system considers the needs of both taxi drivers and riders.

Taxi dispatching services [13], [14], [15], [16], [17] usually send a taxi close to a passenger as per the passenger's call without considering taxi-sharing. Consequently, only vacant taxis need to be examined for each dispatch, which can be easily retrieved by answering a range query. In our case, each taxi that is occupied under full capacity needs to be considered. This complication introduces new challenges.

7.2 Carpool and Dial-A-Ride

Carpool often refers to ridesharing which deals with routine commutes. There are already websites and mobile Apps for this purpose, such as Avego. Given the usual small size of

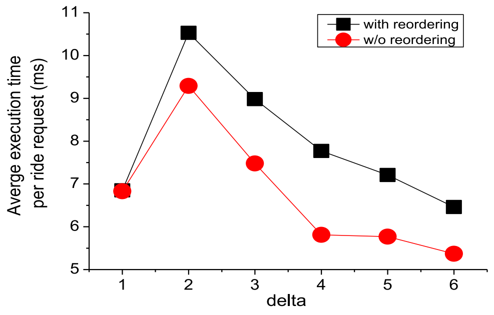


Fig. 19. Time cost of schedule reordering.

the problem, researchers are able to solve it optimally by using linear programming techniques [1], [2]. Unlike in carpool where ride requests are known in advance, the real-time taxi-sharing system here is more challenging as ride requests are generated on-the-fly and the routes of taxis change continuously.

The taxi-sharing problem can be viewed as a special member of the general class of the dial-a-ride problem (DARP). The DARP is originated from and has been studied in various transport scenarios, notably goods transport [18], paratransit for handicapped and elderly personnel [19], etc. Existing works on the DARP have primarily focused on the static DARP, where all customer ride requests are known in priori. Since the general DARP is NP-hard, only small instances (involving only a few cars and dozens of ride requests) can be solved optimally (often by resorting to integer programming techniques, see [20], [21]). Large static DARP instances are usually solved by using the two-phase scheduling strategy [6], [22], [23], [24] with heuristics. Specifically, the phase I partitions ride requests into some group and computes an initial schedule for delivering the riders in each group. In phase II, ride requests are swapped between different groups, aiming to find new schedules optimizing a predefined objective function.

Little research has been carried out on the dynamic DARP, where requests are generated on the fly. Previous works on the dynamic DARP problem [25] continues to adapt the two-phase scheduling strategy. However, the two-phase strategy is not feasible for the real-time taxi-sharing. As if the strategy is applied, the cloud will not serve a new request immediately. Instead it needs to wait for more requests in order to make the phase II possible, which prolongs the response time of a request. In addition, the heavy computation load of the phase II will further increase the response time, resulting in many requests unsatisfied.

7.3 Real-Time Taxi-Sharing

Though real-time taxi-sharing has been studied in several previous works [4], [26], [27], [28], [29], our work demonstrates three major advantages. First, our problem definition is more realistic by considering three different types of constraints. Some existing works (e.g., [28]) did not consider time window constraints and none of these previous works explicitly modelled monetary constraints. (though [30] implicitly considers monetary constraint by converting it to distance constraint.) Second, we analysed the computational cost of each component of the system, proposing a spatio-temporal index and a taxi searching algorithm, which significantly improve the system efficiency. Third, simulation results presented here is more convincing as we evaluated our system based on the real data and at a much larger scale than most previous works did. This is also one of main concepts of urban computing [31], which tackles the big challenges in cities by using big data. The size of the ride request stream in our experiment is as large as 20K and these ride requests are learned from the historical trajectory data set. Detail comparisons between this work and our previous paper [3] are stated in the end of Section 1.

8 Conclusions

This paper proposed and developed a mobile-cloud based real-time taxi-sharing system. We presented detail interactions between end users (i.e. taxi riders and drivers) and the Cloud. We validated our system based on a GPS trajectory data set generated by 33,000 taxis over three months, in which over 10 million ride requests were extracted. The experimental results demonstrated the effectiveness and efficiency of our system in serving real-time ride requests. Firstly, our system can enhance the delivery capability of taxis in a city so as to satisfy the commute of more people. For instance, when the ratio between the number of taxi ride requests and the number of taxis is 6, our proposed system served three times as many ride requests as that with no taxi-sharing. Secondly, the system saves the total travel distance of taxis when delivering passengers, e.g., it saved 11 percent travel distance with the same ratio mentioned above. Supposing a taxi consumes 8 liters of gasoline per 100 km and given the fact learned from the real trajectory data set that the average travel distance of a taxi in a day in Beijing is about 480 km, the system can save over one third million liter of gasoline per day, which is over 120 million liter of gasoline per year (worth about 150 million dollar). Thirdly, the system can also save the taxi fare for each individual rider while the profit of taxi drivers does not decrease compared with the case where no taxi-sharing is conducted. Using the proposed monetary constraints, the system guarantees that any rider that participates in taxi-sharing saves 7 percent fare on average. In addition, the experimental results justified the importance of the dual-side searching algorithm. Compared to the single-side taxi searching algorithm, the dual-side taxi searching algorithm reduced the computation cost by over 50 percent, while the travel distance was only about 1 percent higher on average. The experimental results also suggest that reordering the points of a schedule before the insertion of the new ride request is not necessary in practice for the purpose of travel distance minimization.

In the future, we consider incorporating the creditability of taxi drivers and riders into the taxi searching and scheduling algorithms. Additionally, we will further reduce the travel distance of taxis via ridesharing. We also consider refining the ridesharing model by introducing social constraints, such as gender preference, habits preference (e.g., some people may prefer co-passengers who do not smoke).

Acknowledgments

This research was partially supported by US Department of Transportation National University Rail Center (NURAIL); Illinois Department of Transportation (METSI); and US National Science Foundation grants IIS-1213013, CCF-1216096, DGE-0549489, IIP-1315169. The work was done in Microsoft Research when the first author was an intern under the supervision of the second author.

References

[1] R. Baldacci, V. Maniezzo, and A. Mingozzi, "An exact method for the car pooling problem based on lagrangean column generation," *Oper. Res.*, vol. 52, no. 3, pp. 422–439, 2004.

[2] R. W. Calvo, F. de Luigi, P. Haastrup, and V. Maniezzo, "A distributed geographic information system for the daily carpooling problem," *Comput. Oper. Res.*, vol. 31, pp. 2263–2278, 2004.

[3] S. Ma, Y. Zheng, and O. Wolfson, "T-Share: A large-scale dynamic ridesharing service," in *Proc. 29th IEEE Int. Conf. Data Eng.*, 2013, pp. 410–421.

[4] E. Kamar and E. Horvitz, "Collaboration and shared plans in the open world: Studies of ridesharing," in *Proc. 21st Int. Jont Conf. Artif. Intell.*, 2009, pp. 187–194.

[5] K. Wong, I. Bell, and G. H. Michael, "Solution of the dial-a-ride problem with multi-dimensional capacity constraints," *Int. Trans. Oper. Res.*, vol. 13, no. 3, pp. 195–208, May 2006.

[6] Z. Xiang, C. Chu, and H. Chen, "A fast heuristic for solving a large-scale static dial-a-ride problem under complex constraints," *Eur. J. Oper. Res.*, vol. 174, no. 2, pp. 1117–1139, 2006.

[7] J. Yuan, Y. Zheng, C. Zhang, W. Xie, X. Xie, G. Sun, and Y. Huang, "T-drive: Driving directions based on taxi trajectories," in *Proc. 18th SIGSPATIAL Int. Conf. Adv. Geographic Inf. Syst.*, 2010, pp. 99–108.

[8] J. Yuan, Y. Zheng, X. Xie, and G. Sun, "Driving with knowledge from the physical world," in *Proc. 17th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, 2011, pp. 316–324.

[9] O. Wolfson, A. P. Sistla, B. Xu, J. Zhou, S. Chamberlain, Y. Yesha, and N. Rishe, "Tracking moving objects using database technology in DOMINO," in *Proc. 4th Int. Workshop Next Generation Inf. Technol. Syst.*, 1999, pp. 112–119.

[10] J. Yuan, Y. Zheng, C. Zhang, X. Xie, and G.-Z. Sun, "An interactive-voting based map matching algorithm," in *Proc. 11th Int. Conf. Mobile Data Manage.*, 2010, pp. 43–52.

[11] J. Yuan, Y. Zheng, L. Zhang, Xi. Xie, and G. Sun, "Where to find my next passenger," in *Proc. 13th Int. Conf. Ubiquitous Comput.*, 2011, pp. 109–118.

[12] Y. Ge, H. Xiong, A. Tuzhilin, K. Xiao, M. Gruteser, and M. Pazzani, "An energy-efficient mobile recommender system," in *Proc. 16th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, 2010, pp. 899–908.

[13] R. Balan, K. Nguyen, and L. Jiang, "Real-time trip information service for a large taxi fleet," in *Proc. 9th Int. Conf. Mob. Syst. Appl. Serv.*, 2011, pp. 99–112.

[14] K. Yamamoto, K. Uesugi, and T. Watanabe, "Adaptive routing of cruising taxis by mutual exchange of pathways," in *Proc. 12th Int. Conf. Knowledge-Based Intell. Inf. Eng. Syst., Part II*, 2008, pp. 559–566.

[15] D. Santani, R. K. Balan, and C. J. Woodard, "Spatio-temporal efficiency in a taxi dispatch system," Research Collection School Of Information Systems, Singapore Management University, Oct. 2008.

[16] D. Zhang and T. He, "CallCab: A unified recommendation system for carpooling and regular taxicab services," in *Proc. IEEE Int. Conf. Big Data*, 2013, pp. 439–447.

[17] W. Wu, W. S. Ng, S. Krishnaswamy, and A. Sinha, "To Taxi or Not to Taxi?—Enabling personalised and real-time transportation decisions for mobile users," in *Proc. IEEE 13th Int. Conf. Mob. Data Manage.*, Jul. 2012, pp. 320–323.

[18] Y. Dumas, J. Desrosiers, and F. Soumis, "The pickup and delivery problem with time windows," *Eur. J. Oper. Res.*, vol. 54, no. 1, pp. 7–22, Sep. 1991.

[19] A. Beaudry, G. Laporte, T. Melo, and S. Nickel, "Dynamic transportation of patients in hospitals," *OR Spectr.*, vol. 32, no. 1, pp. 77–107, 2010.

[20] J. Cordeau, "A branch-and-cut algorithm for the dial-a-ride problem," *Oper. Res.*, vol. 54, pp. 573–586, 2003.

[21] L. M. Hvattum, A. Løkketangen, and G. Laporte, "A branch-and-regret heuristic for stochastic and dynamic vehicle routing problems," *Networks*, vol. 49, no. 4, pp. 330–340, Jul. 2007.

[22] J.-F. Cordeau and G. Laporte, "The dial-a-ride problem: Models and algorithms," *Ann. Oper. Res.*, vol. 153, no. 1, 2007.

[23] C. J.-F. and L. G., "A tabu search heuristic for the static multi-vehicle dial-a-ride problem," *Transp. Res. Part B Methodol.*, vol. 37, no. 6, pp. 579–594, 2003.

[24] A. Attanasio, J.-F. Cordeau, G. Ghiani, and G. Laporte, "Parallel Tabu search heuristics for the dynamic multi-vehicle dial-a-ride problem," *Parallel Comput.*, vol. 30, no. 3, pp. 377–387, Mar. 2004.

[25] M. E. T. Horn, "Fleet scheduling and dispatching for demand-responsive passenger services," *Transp. Res. Part C Emerg. Technol.*, vol. 10, no. 1, pp. 35–63, 2002.

[26] P.-Y. Chen, J.-W. Liu, and W.-T. Chen, "A fuel-saving and pollution-reducing dynamic taxi-sharing protocol in VANETs," in *Proc. IEEE 72nd Veh. Technol. Conf.*, Sep. 2010, pp. 1–5.

[27] P. M. d'Orey, R. Fernandes, and M. Ferreira, "Empirical evaluation of a dynamic and distributed taxi-sharing system," in *Proc. 15th Int. IEEE Conf. Intell. Transp. Syst.*, Sep. 2012, pp. 140–146.

[28] G. Gidofalvi, T. B. Pedersen, T. Risch, and E. Zeitler, "Highly scalable trip grouping for large-scale collective transportation systems," in *Proc. 11th Int. Conf. Extending Database Technol.: Adv. Database Technol.*, 2008, pp. 678–689.

[29] S. Ma and O. Wolfson, "Analysis and evaluation of the slugging form of ridesharing," in *Proc. 21st ACM SIGSPATIAL Int. Conf. Adv. Geographic Inf. Syst.*, 2013, pp. 64–73.

[30] G. Gidofalvi and T. Pedersen, "Cab-sharing: An effective, door-to-door, on-demand transportation service," in *Proc. 6th Eur. Congr. Intell. Transp. Syst. Serv.*, 2007.

[31] Y. Zheng, L. Capra, O. Wolfson, and H. Y., "Urban Computing: Concepts, methodologies, and applications," *ACM Trans. Intell. Syst. Technol.*, vol. 5, no. 3, article 38, Sep. 2014.

[32] Y. Wang, Y. Zheng, and Y. Xue, "Travel time estimation of a path using sparse trajectories," in *Proc. 20th ACM SIGKDD (KDD '14)*, ACM, New York, NY, USA, 2014, pp. 25–34.

Shuo Ma received the BS degree in computer science in 2008 from Beijing University of Posts and Telecommunications, Beijing, China. He is currently working toward the PhD degree with the Department of Computer Science, University of Illinois at Chicago. His research interests include mobile computing, computational transportation science, and spatial databases.

Yu Zheng is a lead researcher from Microsoft Research. He is also a visiting chair professor at Shanghai Jiao Tong University and a visiting professor at Southwest Jiaotong University. He is passionate about using big and heterogeneous data generated in urban spaces to tackle the big challenges that cities face, such as transportation, pollution, and energy. He has published more than 50 referred papers as a leading author at prestigious conferences and journals, such as SIGMOD, *VLDB*, *ICDE*, SIGKDD, UbiComp, and *IEEE TKDE*, where he received four best paper awards, e.g., the best paper runner-up award at ICDE 2013. He is currently a member of Editorial Advisory Board of *IEEE Spectrum* and has been serving over 10 prestigious international conferences as a chair, such as the program chair of ICDE 2014, a demo chair of SSTD 2013, and a program chair of UIC 2014. He has been invited to give over 10 keynote speeches at international conferences and forums, e.g., IE 2014 and APEC 2014 smart city forum. In 2013, he was named Top Innovators under 35 by MIT Technology Review (TR35) for his research using data sciences to solve urban challenges. He is a senior member of the IEEE and a senior member of the ACM.

Ouri Wolfson (F' 12) received the BA degree in mathematics and the PhD degree in computer science from the Courant Institute of Mathematical Sciences, New York University, New York, NY. He was a consultant with Argonne National Laboratory, Argonne, IL; the U.S. Army Research Laboratory, Adelphi, MD; the Defense Advanced Research Projects Agency, Arlington, VA; and the Center of Excellence in Space Data and Information Sciences, Goddard Space Flight Center, National Aeronautics and Space Administration, Greenbelt, MD. He was also with the computer science faculty of the Technion—Israel Institute of Technology, Haifa, Israel, and Columbia University, New York. He has been a member of Technical Staff with Bell Laboratories. He is currently the Richard and Loan Hill professor of computer science with the University of Illinois at Chicago and an affiliate professor with the Department of Computer Science, University of Illinois at Urbana-Champaign. His research interests include database systems, distributed systems, and mobile/pervasive computing. He founded Mobitrac, which is a high-tech startup that was acquired by Fluensee Co. in 2006. He is a fellow of the Association for Computing Machinery, the American Association for the Advancement of Science, and the IEEE.