QA-Share: Towards Efficient QoS-Aware Dispatching Approach for Urban Taxi-sharing

Shanfeng Zhang[1,2], Qiang Ma[2], Yanyong Zhang[3], Kebin Liu[2], Tong Zhu[2], Yunhao Liu[2]
1 Department of Computer Science and Engineering, Hong Kong University of Science and Technology
2 School of Software and TNList, Tsinghua University, Beijing, China
3 Department of Electrical and Computer Engineering, Rutgers University, USA
{shanfeng, maq, yanyong, kebin, tong, yunhao}@greenorbs.com

***Abstract*—Taxi-sharing allows occupied taxis to pick up new passengers on the fly, promising to reduce waiting time for taxi riders and increase productivity for drivers. However, if not carefully designed, taxi-sharing may cause more harm than benefit – it becomes harder to strike the balance between driver's profit and passenger's quality of service (e.g. travel time, number of strangers that share a taxi, etc.). In this paper, we propose a QoS-aware taxi-sharing system design – QA-Share – by addressing two important challenges. First, QA-Share aims to maximize driver profit and user experience at the same time. Second, QA-Share continuously optimizes these two metrics by dynamically adapting its schedule as new requests arrive, without entering an oscillation state.**

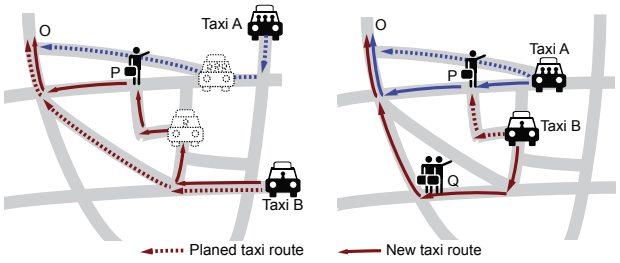
To address these two challenges, we have formulated the optimization problem using integer linear programming, and derived the optimal solution under a small system scale. When the number of requests and taxis becomes large, we have devised a heuristic algorithm that has a much faster execution time. We have also studied how to minimize oscillations caused by schedule re-calculations by dynamically tuning the update threshold. We have evaluated our approach with real-world dataset in a Chinese city – ZhenJiang – which contains the GPS traces recorded by over 3,000 taxis during a period of three months in 2013. Our results show that the QoS and profit is increased by 38% compared to earlier schemes.

I. Introduction

In modern cities, taxis are playing an increasingly important role in supporting people's commute. According to a survey conducted in New York city [1], over 100 taxi companies operate more than 13,000 taxis in New York, delivering 660,000 passengers every day. They convey more than 25% of the passengers, accounting for 45% of total transit fares. Though important, today's taxi system is far from being efficient: a common problem is that people often have difficulties in hailing a taxi during rush hours, while occupied taxis may still have available seats [2]. This inefficiency becomes much worse in countries and areas that are still in the process of urbanization, where the city population increases at a faster rate than the number of taxis.

This problem has attracted some attention in recent years, resulting in a proposed solution called ride-sharing, in which people share a taxi with others who have similar itineraries and schedules. When a new passenger request arrives, the taxi-sharing server will try to dispatch a taxi to satisfy the new request while serving its existing requests (if any). Taxi-sharing has to track passenger requests and taxi locations on a real-time basis, both of which are highly dynamic and hard to predict, thus posing great challenges on the underlying system.

978-1-4673-7331-9/15/31.00 © 2015*IEEE*



(a) Request $R_1(P \rightarrow O)$ is received. Taxi B is better to serve the request.

(b) Then, request $R_2(Q \rightarrow O)$ is received. It's better to update the current schedule.

Fig. 1. Example to show the two challenges

Several taxi-sharing schemes have been proposed in the literature, such as those discussed in [3–6]. However, they all fall short of the requirements of a scalable taxi-sharing system that serves real-time user requests. For example, CoRide [5] focuses on static scenarios in which passengers and taxis are at the same pick-up location (e.g., an airport). When a new request arrives, CoRide simply inserts the request to the end of an occupied taxi's schedule, or arranges an empty taxi. On the other hand, T-Share [6] does consider dynamic scenarios, but its scheduling is rather simplistic: for each new request, it simply assigns the taxi with the minimum additional driving distance, without considering the Quality of Service (QoS) for the passengers on the chosen taxi. Once a sharing schedule is made, it remains unchanged even though a new schedule may be more efficient.

In this study, we aim to improve the state-of-the-art taxi-sharing strategy by addressing two important challenges. The first challenge we target at is to find balance between enhancing QoS for passengers (e.g. travel time, number of strangers that share a taxi, etc.) and maximizing profit for taxi drivers. As an example, in traditional non-sharing taxi services, greedy drivers may choose a longer route to increase his/her own profit, extending the passenger's travel time. In a sharing system, if not careful, passenger's QoS will be hurt even more. People choose to take taxis for a comfortable, door-to-door

service. By sharing a taxi with strangers, passengers may not only feel a loss of privacy, but also have to take a longer route in many cases, which may severely undermine their motivation for participating in taxi-sharing. Fig. 1(a) illustrates our up-mentioned point. As shown a request R_1 arrives with pick-up location P and destination location O. The dashed lines represent the planned routes of taxi A and B. Both CoRide [5] and T-Share [6] choose taxi A to serve the request, because of its shorter detour distance. Clearly, this selection does not take into consideration passengers' QoS preferences – it affects three existing passengers on taxi A, whereas the selection of taxi B will only affect one existing passenger.

Another challenge we target at is the dynamic nature of taxi-sharing, due to dynamically arriving requests. For example, in Fig. 1(b), after scheduling Taxi B to serve request R_1, let us assume that the dispatching system receives an urgent request R_2 from pick-up location Q to the destination O. In this case, taxi A is too far away to serve R_2, and taxi B is going to be too late for R_2 if it has to serve R_1 first. Therefore, a better choice is to dynamically update the schedule such that taxi B directly goes to serve R_2, while taxi A goes to serve R_1. Dynamically updating schedules is very challenging problem, because one needs to deal with a large number of requests, and needs to avoid such situations that taxis keep reversing directions with dynamic updates of schedules.

To address these challenges, in this work, we propose an efficient and scalable QoS-Aware dispatching approach for taxi-sharing, referred to as *QA-Share*. In this paper, QoS refers to the waiting time for a taxi, the traveling time to the destination, and the number of strangers to share the taxi. We propose a taxi-sharing model such that the QoS of passengers and the profit of drivers are maximized at the same time. We formally define the taxi-sharing dispatching problem, which is shown to be NP-hard. We then provide optimal dispatching solutions with integer linear programming for a small number of requests. When the number of requests becomes sufficiently large, we propose a heuristic online algorithm to speed up the scheduling process. We design a map clustering algorithm and build indexes of taxi locations. Based on the indexes, a fast taxi searching and scheduling algorithm is then discussed. We also design a dynamic update mechanism to re-schedule the taxi-sharing on the fly as new requests arrive.

We conduct trace-driven simulation with realistic dataset in a Chinese city – ZhenJiang – which contains the GPS traces recored by over 3,000 taxis during a period of three months in 2013. The detailed evaluation results show that compared with earlier work, the QoS and profit is increased by 38%. Compared with the ground truth, the fraction of requests that get served is increased by up to 200%. To sum up, the key contributions of this work are four-fold:

- To the best of our knowledge, this is the first study that proposes a taxi-sharing service that maximizes the QoS of passengers as well as the profit of drivers;
- We give optimal dispatching solutions by integer linear programming, and propose heuristic algorithms to speed up the process.

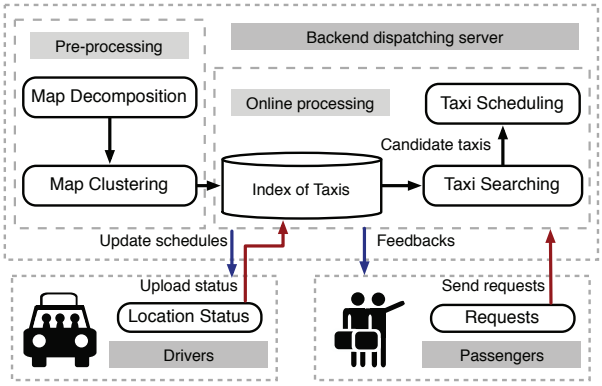


Fig. 2. System architecture

- We use a dynamic update mechanism to further maximize profit and minimize oscillations caused by schedule re-calculations by dynamically tuning the update thresholds;
- This is the first study that conducts simulations with traces from a city with population of 3 million, showing that taxi-sharing is viable in a medium city.

The rest of the paper is organized as follows. Sections II introduces some related work. Section III presents the system overview. Section IV describes the pre-processing stage including map decomposition and clustering. Section V explains the algorithms for taxi scheduling. Section VI presents our update mechanism. Finally Section VII validates our system, followed by the conclusion in Section VIII.

II. System Overview

Fig. 2 sketches the architecture of QA-Share, which consists of three major players: passengers, taxi drivers, and a dispatching server.

A. Passengers

As shown in Fig. 2 (bottom right), passengers send delivery requests to the dispatching server. A delivery request R contains the following attributes: the contact phone number,

TABLE I
Summary of symbols

$G(N, E)$	A road network
R	A passenger request for taxi service
$R.p$	The pick-up location of R
$R.d$	The drop-off location of R
$R.pw$	The pick-up window of R
$R.dt$	The delivery deadline of R
$R.n$	The number of passengers of R
$R.t$	The number of tips of R
$R.f$	The basic ride fare of R
V	A taxi status
$V.s$	The current schedule of V
$V.n$	The number of available seats of V
$V.l$	The current location of V
C	A region in road network
$C.m$	The landmark of C
$C.L_c$	The list of regions nearby C
$C.L_t$	The taxi IDs ordered by the time entering C

number of passengers $R.n$, pick-up location $R.p$, pick-up time window $R.pw$, drop-off location $R.d$ and drop-off deadline $R.dt$. Here pick-up time window $R.pw$ is the interval during which the passenger needs to be picked up. Immediately after submitting a request, the passenger receives feedback from the server, including the estimated pick-up time, estimated drop-off time, and estimated fee. The estimated fee may not be the final price. For example, when the passenger share the taxi with further more passengers, the price will decrease. Please note that, we use the algorithm proposed in [7] to estimate travel time between two locations, which has an average estimation error of 0.4 min/km. Based on the travel distance and average estimation error, we calculate the overall error ϵ. We guarantee that the passengers are delivered before $R.dt - \epsilon$. For simplicity, we use $R.dt$ to denote the updated delivery deadline.

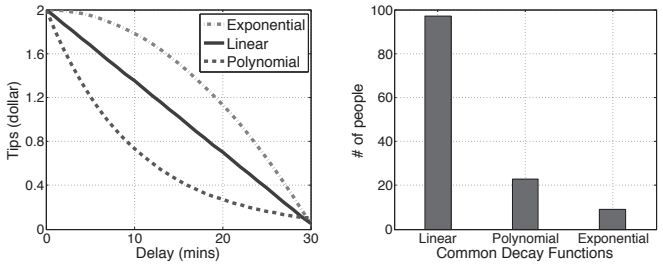
QA-Share also accepts requests that prefer exclusive taxi usage, and at the same time, it encourages taxi sharing through an attractive pricing mode. In our model, taxi fare consists of two parts: the basic taxi fare $R.f$ and the dynamic taxi fare denoted by tips $R.t$, which we explain in detail below.

Basic taxi fare: Based on the existing taxi-sharing pricing scheme and T-Share [6], the basic taxi fare in our model, if participating in taxi-sharing, is ρf_0, where f_0 is the taxi fare precomputed by multiplying unit price and the distance of the shortest path to deliver the passenger, and ρ is the fare discount parameter, usually set as 0.6 [8]. Compared with traditional non-sharing pricing scheme that is based on the travel distance of the taxi, our model ensures that drivers won't make unnecessary detours to make profit.

Tips: In developed countries such as US, people often pay tips for taxi services, while people in developing countries have rarely paid tips before. Recently, as it has been increasingly difficult to hail a taxi, passengers are more inclined to pay tips to attract a taxi. For example, in Beijing many people start to offer tips through taxi hailing apps.

In QA-Share, passengers are encouraged to pay tips according to their QoS requirements, such as how much extra delay one can tolerate, the maximum number of people one is willing to share a taxi with, etc. In this paper, we focus our discussion on travel delay. The number of passengers can be somehow inferred in travel delay: the more passengers with different itineraries, the more travel delay each individual passenger may experience. Please note that we can easily extend our work to consider more QoS requirements.

As far as delay is concerned, tip amount is a decay function of the delay. Passengers can choose a decay function by themselves. Exponential decay ($R.t = \alpha e^{-\lambda delay}$), polynomial decay ($R.t = \alpha - \beta * delay^{\lambda}$) and linear decay ($R.t = \alpha - \beta * delay$) are common functions to model the decaying process. Fig. 3(a) shows a few example decay functions. In this study, we surveyed 129 people about their choice of decay function, and found that 107 people chose the linear decay function due to its simplicity. As a result, we will then focus on the linear function in the remaining of this paper. In the linear decay function, the first parameter is the maximum tip they are willing to pay (when the taxi does not cause any delay), and the second parameter is the deduction amount when the passenger is delayed by one minute. Please note that passengers usually do not pay a tip more than $(1 - \rho)f_0$, in which case, they would rather choose to ride alone.



(a) Example of three different decay functions.

(b) Most people prefer a simple linear decay function.

Fig. 3. Comparison of three common decay functions

B. Drivers

As shown in Fig. 2 (bottom left), taxi drivers upload his/her GPS data to the server with a fixed rate, and receive new driving routes from the server when a route update is available. As smart phones are ubiquitous nowadays, and on-vehicle energy is available to charge phones, we assume drivers can use smart phones to interact with the server. We also assume the taxi drivers follow the driving routes scheduled by the dispatching server because our dispatching strategy aims to maximize the expected profit of the drivers.

C. Server

As shown in Fig. 2 (top), the backend dispatching server is mainly in charge of 1) receiving requests from passengers, 2) obtaining taxi status updates, 3) scheduling routes to serve the passengers, and 4) estimating fares for passengers. The server's processing has two stages: preprocessing and online processing.

Pre-processing stage: Considering the difficulty of directly working with a stream of $2D$ geographic data, we first represent the city using a graph $G(N, E)$ in the preprocessing stage, in which a node represents a location with latitude/longitude coordinate and an edge represents a path between two locations (e.g. a street). A long street can be represented by several edges. As such, a taxi's trajectory is represented by a series of nodes/edges.

Online processing stage: In the online processing stage, the server tracks each taxi's trajectory, and processes new requests every PW minutes such that the weighted sum of QoS as well as the sum of profit are both maximized. If the number of requests is small, we formulate the problem as an integer linear programming problem, and derive the optimal solution. However, this approach does not scale to a large number of requests. When the request number becomes sufficiently large, we devise a heuristic two-phase scheduling algorithm. In the algorithm, we (1) build an index structure based on

map clustering to facilitate efficient search of candidate taxis, (2) calculate the fare for each candidate taxi, (3) construct a weighted bipartite graph between taxis and requests, and (4) match taxis with requests based on maximum matchings in the bipartite graph.

III. Pre-processing

In the pre-processing stage, we first represent a 2 dimensional city area using a graph $G(N, E)$. We then design a geographic clustering algorithm to divide the graph $G(N, E)$ into k regions, which facilitates fast estimation of travel time between two locations, as well as index construction in the online stage.

A. Map Decomposition

As shown in Fig. 4, we first discretize the terrain of a city, which is a continuous 2-D area, into a graph $G(N, E)$ (a.k.a. digital map), where N represents locations with latitude/longitude coordinates, and E represents paths between any two locations [9]. For each edge, we also estimate the travel time of the corresponding road segment. Several studies [10, 11] looked at estimating traffic conditions on the edges in G. In this paper, we use the regression algorithm proposed in [11] to estimate a static transition speed on each edge. We can easily extend our work to a dynamic traffic scenario by considering the time-dependent trajectory regression algorithms such as the one discussed in [10].

B. Map Clustering

Searching for shortest paths in a weighed graph is rather complex, and we choose to cluster the graph to simplify travel time estimation between any two locations. In this paper, we devise a clustering algorithm to split the road network $G(N, E)$ into separate landmark regions.

Definition 1. (Landmark): A $landmark$ is a node $n \in N$, such that the number of pick-up locations near n is above a threshold. Here we say two locations are near if the travel time between them is smaller than a threshold r.

We first generate k landmarks from past pick-up locations. The main reason to cluster the graph by landmarks is that we can easily find a nearby reference location for pick-up locations, and reduce the expected estimation error. We form a cluster for each landmark, and merge other nodes to the cluster with shortest travel time to the corresponding landmark. Fig. 4 shows a result of clustering a city area, where each color represents a cluster. We associate an estimation error $C_i.err$ for each cluster C_i, which is the longest travel time between C_i's landmark and any node in C_i.

We also precompute the shortest travel time (denoted by T_{ij}) between each landmark pair $C_i.m$ and $C_j.m$. The results are saved in a distance matrix T. Now imagine that each geographical point collapses to the nearest landmark. Given two nodes in clusters C_i and C_j, the estimated travel time t is bound by Equ. 1:

$$|t - T_{ij}| \leq C_i.err + C_j.err. \quad (1)$$

Using this novel travel time estimation approach, we avoid the expensive shortest path calculation.

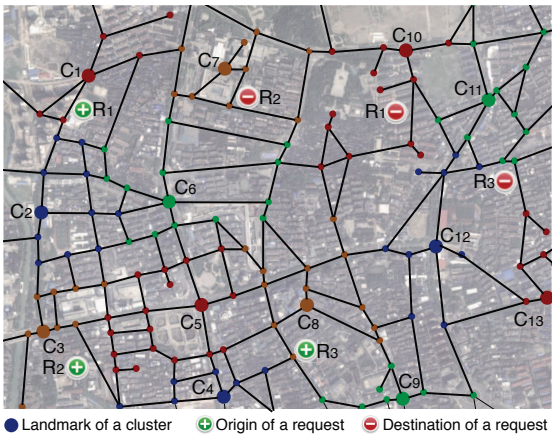


Fig. 4. Map decomposition and clustering of an area with 3 requests

IV. Online processing

In this section, we first give a formal definition of the taxi-sharing problem, which is NP-hard. We then propose an optimal algorithm that is practical for a small number of requests. When the number of requests becomes large, we propose a two-phase heuristic algorithm.

A. Problem Formulation

We use $\mathcal{R} = \{R_1, \cdots, R_n\}$ to denote the set of requests at present, and $\mathcal{V} = \{V_1, \cdots, V_m\}$ the set of on-road taxis. We define the real-time taxi-sharing problem in a complete directed graph $G = (N, A)$, where $N = O \cup P \cup D$, $O = \{1, \cdots, m\}$, $P = \{m+1, \cdots, m+n\}$ and $D = \{m+n+1, \cdots, m+2n\}$. P and D denote pick-up locations $\{R_1.p, \cdots, R_n.p\}$ and drop-off locations $\{R_1.d, \cdots, R_n.d\}$, while O represents the current locations of m taxis $\{V_1.l, \cdots, V_m.l\}$.

We assume each taxi has the same capacity c (i.e., number of seats). Each vertex $i \in N$ is associated with a load q_i, such that $q_i \geq 0$ for $i \in O \cup P$ and $q_i = -q_{i+n}$ for $i \in P$. We use load to represent the number of passengers of a request. The load at a pick-up location is positive, while the load at a drop-off location is negative. A time window $[e_i, l_i]$ is associated with node $i \in P \cup D$, where e_i and l_i represents the earliest and latest time, at which service may begin at node i. For each vertex $i \in D$, we use β_i to denote the tip parameter β of the corresponding request R_{i-m-n}. Each arc $(i, j) \in A$ is associated with an accurate travel time t_{ij}, which is estimated using the travel time estimation algorithm discussed in [12, 7].

For each arc $(i, j) \in A$ and each vehicle $k \in \mathcal{V}$, let $x_{ij}^k = 1$ if vehicle k is scheduled to travel from node i to node j; otherwise x_{ij}^k is 0. For each node $i \in N$ and each vehicle $k \in \mathcal{V}$, let T_i^k be the time when k arrives at node i, and Q_i^k be the load of vehicle k after visiting node i.

This problem is a generalization of the Traveling Salesman Problem with Time Window, which has already been proved to be NP-complete. So our taxi-sharing problem is NP-hard. We thus omit the detailed proof for the sake of space.

B. Optimal Solution

We solve the optimal solution for the above formulation through Integer Linear Programming (ILP).

Next, we introduce the constraints of ILP:

- The flow out of the original location of each taxi is at most one:

$$\sum_{j \in N} x_{k,j}^k \leq 1 \qquad \forall k \in \mathcal{V}. \tag{2}$$

- All the flows entering a pick-up location will eventually leave the location (Equ. 3):

$$\sum_{j \in N} x_{ji}^k = \sum_{j \in N} x_{ij}^k \qquad \forall i \in P, k \in \mathcal{V}. \tag{3}$$

- The flow of all taxis out of a pick-up location is 1 (Equ. 4), which means that only one taxi is scheduled to serve the request. Remember that we create several nodes in the directed graph G, even if several requests share the same pick-up location. The same taxi also traverses the corresponding drop-off location (Equ. 5).

$$\sum_{k \in \mathcal{V}} \sum_{j \in N} x_{ij}^k = 1 \qquad \forall i \in P. \tag{4}$$

$$\sum_{j \in N} x_{ij}^k = \sum_{j \in N} x_{j,i+n}^k \qquad \forall i \in P, k \in \mathcal{V}. \tag{5}$$

- The flow entering a destination is larger than the departing flow:

$$\sum_{j \in N} x_{ji}^k \geq \sum_{j \in N} x_{ij}^k \qquad \forall i \in D, k \in \mathcal{V}. \tag{6}$$

- The consistency of arrival time at nodes is constrained by Equ. 7. The arrival time at each node is in the required interval (Equ. 8), and the arrival time at the pick-up node is earlier than the corresponding drop-off node (Equ. 9).

$$T_j^k \geq (T_i^k + t_{ij}) x_{ij}^k \qquad \forall i \in N, j \in N, k \in \mathcal{V}. \tag{7}$$

$$e_i \leq T_i^k \leq l_i \qquad \forall i \in P \cup D, k \in \mathcal{V}. \tag{8}$$

$$T_i^k \leq T_{n+i}^k \qquad \forall i \in P, k \in \mathcal{V}. \tag{9}$$

- We constrain the number passengers on the taxi (Equ. 10), which is not to exceed the taxi capacity (Equ. 11).

$$Q_j^k = (Q_i^k + q_j) x_{ij}^k \qquad \forall i \in N, j \in N, k \in \mathcal{V}. \tag{10}$$

$$Q_i^k \leq c \qquad \forall i \in N, k \in \mathcal{V}. \tag{11}$$

Equ. 7 and 10 are not linear. Inspired by [13], we introduce constants M_{ij}^k and W_{ij}^k such that these constraints can be made linear:

$$T_j^k \geq T_i^k + t_{ij} - M_{ij}^k (1 - x_{ij}^k) \; \forall i \in V, j \in V, k \in K, \tag{12}$$

$$Q_j^k = Q_i^k + q_j - W_{ij}^k (1 - x_{ij}^k) \; \forall i \in V, j \in V, k \in K. \tag{13}$$

These constraints are validated by setting $M_{ij}^k \geq max\{0, l_i + t_{ij} - e_j\}$ and $W_{ij}^k \geq min\{c, c + q_i\}$

In the taxi-sharing problem, we have two objective functions. The first one is to maximize the profit of the drivers. The profit consists of basic taxi fare and the tips. As discussed in Sec. II-A, the basic taxi fare is same when the request is served by different drivers or with different traveling time. Therefore, we only need to maximize the tips:

$$\max \sum_{i \in D} \Big(\alpha_i - \beta_i (\sum_k T_i^k - e_i) \Big). \tag{14}$$

The second objective is to maximize the QoS of passengers, which is quantified as the sum of weighted delay time of passengers:

$$\min \sum_{i \in D} \beta_i \sum_{k \in \mathcal{V}} T_i^k. \tag{15}$$

As α_i and e_i are constant in the problem model, these two objective functions are consistent with each other. In other words, we could maximize the profit of drivers and the QoS of passengers at the same time. For simplicity, we use profit to quantify the QoS in the following part of this paper.

When we have a small number of requests and taxis, we can solve this ILP formulation with a branch and cut algorithm.

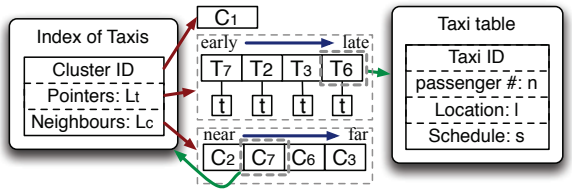


Fig. 5. Index of taxis

C. Heuristic Taxi-Sharing Algorithm

Our dispatching system aims to respond to requests within a short delay. Solving the proposed ILP problem is too time-consuming when we have a large number of requests, thus rendering the system unable to serve the requests on the fly. In this case, instead of obtaining the global optimal schedule, we propose a heuristic approach that reduces the problem space by focusing on the taxis in the requested region. In order to locate taxis in a certain location, we need to build efficient index structures to facilitate location-based taxi search.

Index: To facilitate fast taxi search, we build an index structure based on the pre-computed clusters. Specifically, for each cluster C_i, we maintain a nearby taxi list $C_i.L_t$ and nearby cluster list $C_i.L_c$, as illustrated in Fig. 5. Here, $C_i.L_c$ consists of a list of nearby clusters, sorted in ascending order of the travel time from C_i. This list is static, and can be pre-computed. The taxi list $C_i.L_t$ records the IDs of taxis that are to arrive C_i in the near future, as well as the estimated arrival time. The taxis in $C_i.L_t$ are sorted in the ascending order of the arrival times, and this list is updated dynamically.

Candidate Taxi Search Algorithm: Next we describe our taxi search algorithm, which is to find a set of candidate taxis that are suit to serve the standing requests – candidates should be able to pick up the passenger(s) within the pick-up window, and have enough available seats to hold the passengers. Further, the candidate's existing requests should still meet their deadlines. The pseudo-code of the proposed algorithm is described in Algo. 1. For the clarity of description,

we explain the algorithm using the example illustrated in Fig. 4, in which we assume the dispatching server receives three requests at the same time, namely R_1, R_2 and R_3. The search algorithm checks each request individually. Taking R_1 for example, we first find the resident cluster C_1, where $R_1.p$ is located. Next we search for candidate clusters. A cluster C_i is considered a *candidate cluster* if taxis located in C_i can arrive at cluster C_1 within the pick-up window:

$$t_{curr} + T_{1i} - C_1.err - C_i.err \leq R_1.pw.l, \qquad (16)$$

where t_{curr} is the current time. The algorithm checks all clusters in the ordered list $C.t$, and breaks when the next cluster fails to satisfy Eqn 16. For the example in Fig. 4, clusters C_2, C_6, C_7 are candidate clusters.

The algorithm next examines each candidate cluster C_i to find out whether there are taxis that satisfy the following two conditions: 1) the arrival time at C_i is no later than $R_1.pw.l - T_{1i} + C_1.err + C_i.err$; and 2) the taxi has enough empty seats when arriving in cluster C_i. As such, we can locate a list of candidate taxis for each request.

QA-Share Scheduling: After identifying candidates for each request, the scheduling algorithm next calculates the schedule for these taxis, with the objective of maximizing drivers' profit and passengers' QoS at the same time.

For each request, as in [6], we assume that the order of pick-up/drop-off locations in the existing schedule remain unchanged while new locations are inserted into the schedule. That is, the scheduling algorithm tries to insert the new request's pick-up and drop-off locations into the taxi's delivery schedule while satisfying the following conditions: 1) the requesting passenger(s) can be picked up and delivered before the specified deadlines; 2) there are enough seats to take the requesting passengers at the pick-up location; and 3) accepting the request increases the profit of the taxi driver. After examining these conditions with each candidate taxi, we pick the one that gives the most profit.

Algorithm 1 Candidate Taxi Searching Algorithm

Input: A road network $G(N, E)$, indexes of taxis $C.L_c$ and $C.L_t$, requests $\mathcal{R}$, and the distance matrix T.
Output: Candidate taxis to serve each request R

```
1:  foreach R ∈ R do
2:      Find resident cluster C_i for R.p;
3:      foreach taxi V ∈ C_i.L_t do
4:          if V arrive C_i before request deadline R.pw.l
            and V.n > R.n then
5:              Mark V as candidate for R;
6:      foreach C_j ∈ C_i.L_c do
7:          if T_ij > R.pw.l then
8:              Break;
9:          foreach V ∈ C_j.L_t do
10:             if t_curr + T_ij − C_i.err − C_j.err < R.pw.l
                and V.n > R.n then
11:                 Mark V as candidate for R;
```

It may so happen that a single taxi can be chosen by multiple requests (that are submitted within a short time frame PW), in which we say a scheduling conflict has occurred. A conflict can be potentially addressed in different ways. In the simplest case, we may be able to schedule the candidate taxi to serve all the requests. If that is impossible, we keep a subset of requests to the chosen taxi, and schedule the rest of requests to the other qualified taxis. We achieve this objective by solving a weighted max bipartite matching problem. In the bipartite graph, one set of nodes are all the requests, while the other set consists of the candidate taxis. An edge between taxi t and request r means t can serve r, where the weight of the edge is the profit if t serves r. We can solve the maximum weighted bipartite matching problem optimally with *Hopcroft-Karp* algorithm [14] with $O(\sqrt{N}E)$ complexity. When we have a large number of nodes, but a small number of edges, this algorithm runs very fast.

D. Dynamic Schedule Update

As discussed in Sec. I, when more requests are submitted, the dispatching system may need to recalculate the schedules. In QA-Share, we adopt the following update mechanism: we periodically re-calculate the schedule using the above method, considering both new requests and those requests that are scheduled but not served. Then we update the schedule if it leads to an increased profit.

With periodic scheduling updates, one may fear that oscillation might occur – a taxi may be requested to drive towards a different direction after each update, leading to no real progress. In order to mitigate oscillation, we propose to update the schedule only when the improved profit is above a threshold Φ. A large Φ value can effectively mitigate the oscillation likelihood. Indeed, as shown in Sec. V, we did not notice much oscillation with $\Phi = 1.5$ dollar/request. In real world taxi-sharing system, we should let the passengers decide the value of Φ, indicate whether they tolerates oscillations or not, and in case they tolerates them how much would they request to be compensated.

V. Evaluation

A. Dataset Description

In our evaluation, we perform a large scale trace-driven simulation using real-world taxi trajectory dataset. The dataset contains more than 81 million GPS locations of over 3,000 taxis during a period of three month, from October 2013 to December 2013, in Zhenjiang (a city in Jiangsu, China). The dataset has the following key attributes: GPS coordinates and taxi status (occupied or not). Based upon the status information, we generate passenger requests – a request's pick-up and drop-off locations are the beginning and end locations of an occupied period. In total, we generated 1.8 million requests.

B. Evaluation Methodology

1) Taxi-sharing Schemes: we compare four dynamic taxi-sharing algorithms: 1) the optimal integer linear programming algorithm, referred to as **Optimal-Share**; 2) the heuristic

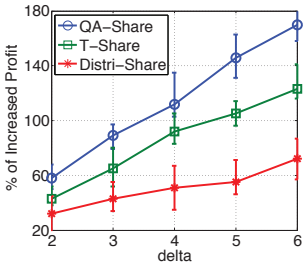


Fig. 6. Profit vs. delta in busy hours

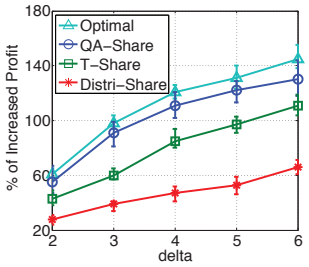


Fig. 7. Profit vs. delta in leisure hours

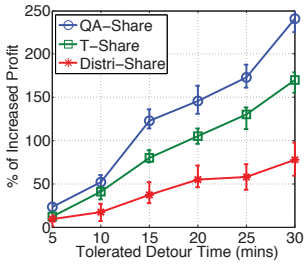


Fig. 8. Profit vs. DT in busy hours

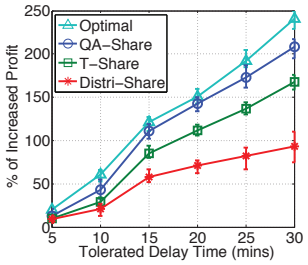


Fig. 9. Profit vs. DT in leisure hours

algorithm, referred to as **QA-Share**; 3)**T-Share** discussed in [6]; and 4) a completely distributed taxi sharing service without a centralized dispatching server, referred to as **Distri-Share**, in which a random taxicab near the request is going to serve the request. In our evaluation, the ground truth is the original GPS traces of the dataset.

2) Simulation Initialization: To make simulations more realistic, we use the real taxi status data in our simulations. Suppose we start simulations from a certain timestamp t_s (the simulated time). We then use each taxicab's information in the trace before t_s as the initial status of that taxi. Specifically, we start each single simulation at 5PM and end at 8PM for rush hour simulations. Most people get off work at this time period and have a large demand of taxis.

3) Simulation Scale: Note that the taxi GPS dataset only includes requests that got served, while many requests were unserved in real life due to failure to get a taxi. The motivation of a taxi-sharing system is to serve these requests that can't be served in traditional taxi system. In order to accommodate the unserved requests in the simulation, we introduce a request inflation ratio δ to simulate more requests than available in the dataset – in our simulations, we combine the requests during δ days in the dataset as the requests simulated in one day. In the dataset, the number of requests in a single day is about 20,000 on average, and we have $\delta = 5$ unless otherwise specified.

4) Request Attributes: Note that we can only extract $R.p$ and $R.d$ from the dataset. We set the earliest pick-up time $R.pw.e$ as the arrival time of the request, the latest pick-up time $R.pw.l$ as $R.pw.e + DT$ where DT is a tolerable detour time; the delivery deadline $R.dt$ as the sum of $R.pw.l$ and the average travel time between the pick-up and drop-off locations; the number of passengers $R.n$ as 1; the tip as a linear decay function, where $R.t.\alpha$ is a random number between 0.5 dollar and 10 dollars; and $R.t.\beta$ calculated by $R.t.\alpha/DT$, where DT is 20 minutes unless otherwise specified.

5) System parameters: We have the following three system parameters: process time window (or schedule update interval) PW (Sec. II-C), schedule update threshold Φ (Sec. IV-D), and the threshold τ that is used to determine whether to use the optimal ILP scheduling or the heuristic approach. Unless otherwise specified, we have $PW = 3min, \Phi = 1.5, \tau = 4$.

C. Simulation Results

Profit: We compare the increased profit with respect to the ground truth (non-sharing profit) by varying the value of δ with the tolerated detour time (DT) of 20 minutes. We used the dataset in busy hours to evaluate $QA-Share$, and the dataset in leisure hours (i.e., 1AM to 5AM) to evaluate $Optimal-Share$ (237 taxicabs).

Fig. 6 reports the percentage of increased profit of QA-share, T-share, and Distri-Share with increasing request rate. We have the following observations. First, as the number of requests increases, the percentage of increased profit for all three schemes also increases, demonstrating that taxi-sharing is a viable solution compared to traditional non-sharing system. Second, we find that it is beneficial to have a centralized dispatching server in taxi-sharing. Third, QA-Share results in higher profit than T-Share, and the improvement can be as high as 37.8%. Fig. 7 evaluates different taxi-sharing schemes in leisure hours, and we observe that QA-Share is close to Optimal-Share, which is the best among all the schemes. The improvement compared with T-Share can be up to 30.2%.

Fig. 8 shows the percentage of increased profit of different sharing schemes with different DT values during busy hours. Here, we observe the same trend as in Fig. 6, with QA-Share leading the performance. The improvement over T-Share can be up to 42.3%. Fig. 9 compares the sharing schemes in leisure hours while varying the DT value. Again, we observe the same trend as in Fig. 7. The improvement of QA-Share over T-Share is up to 27.7%.

Request Service Rate is the fraction of requests that were served in the simulations. We first compare the service rates by varying the value of δ during the busy hours, and summarize the results in Fig. 10. We observe that as the number of requests increases, the service rate decreases, and QA-share performs better than T-share by as much as 28.9%. We also compare the service rate by varying the passengers' tolerable detour time (with $\delta = 4$) and show the results in Fig. 11. As expected, the service rate increases when the value of DT increases. In particular, QA-share performs much better than T-share at larger DT values, with an improvement of 25.8%. This is because with larger DT values, the requests are more likely to be rescheduled for better performance.

D. Evaluation of system parameters

Process Window(PW): QA-share re-calculates the schedule every PW time. We compare the profit gain by varying

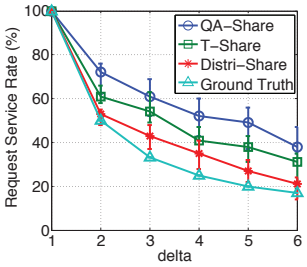


Fig. 10. Service rate vs. delta

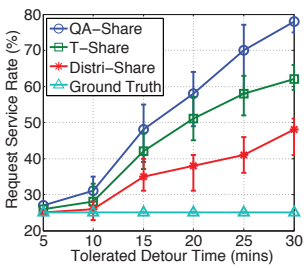


Fig. 11. Service rate vs. DT

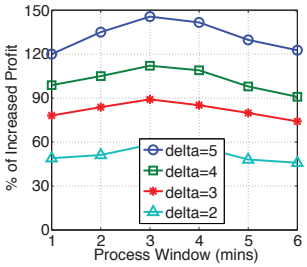


Fig. 12. Profit vs. Process Window

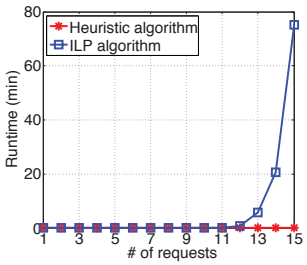


Fig. 13. Running time comparison

the value of PW. We set $DT = 20\ mins$ in this set of experiments. Fig. 12 shows the simulation results. We find that QA-Share achieves the maximum profit gain when PW is between 2 and 5. This is because large process windows lead to delayed processing of the requests; a small process window only leads to a small number of requests to be considered, hence less performance gain.

Algorithm threshold(τ): This threshold is used to decide whether to use QA-Share or Optimal-Share when calculating the schedule for a single round. We seek to find a suitable value for τ by observing the execution time of Optimal-Share. Fig. 13 shows the results. We observe that the execution time of Optimal-Share increases considerably when the number of requests is larger than 12. In this case, we can set $\tau = 12$.

Update threshold(Φ): We update the schedules if the new schedule has a profit increase larger than Φ. Here, we evaluate the impact of Φ on profit gain, and the degree of oscillations. An oscillation is defined as a taxi travel to the same location three times in 5 minutes. Fig. 16 shows the simulation results. With the increase of Φ, the percentage of oscillation decreases. The profit gain reaches the maximum when we have $\Phi = 1.5$. Low update thresholds result in oscillations, while high update thresholds lead to little change to the schedule. We note that even though an update threshold at 1.5 leads to maximum profit, it also results in 3.2% of oscillations. In order to experience a lower oscillation likelihood, we recommend a slightly higher threshold value.

VI. Related Work

Resource allocation has been well studied [15–17]. Allocating taxis, however, is different and attracts only a few studies. We review the state-of-the-art work in three categories: taxi-sharing systems, carpooling and taxi dispatching systems.

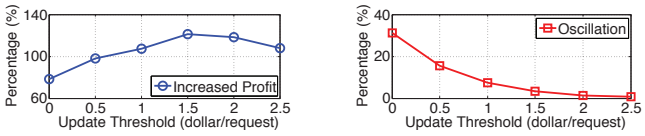


Fig. 14. Influence of update threshold

A. Taxi-sharing Systems

A number of papers [3–6] address the dispatching problem for taxi-sharing. However, they all fall short of the requirements of a scalable taxi-sharing system that serves real-time user requests. Some of them dispatch taxis based on simple and insufficient constraints, e.g. the taxi capacity [4], and the passenger's preference of gender to share a taxi [3]. Some other studies [5, 6] consider more practical and general constraints including the pick-up and drop-off time requirement. CoRide [5] focuses on static scenarios, in which passengers and taxis are at the same pick-up location (e.g., an airport). When a new request arrives, CoRide simply inserts the request to the end of an occupied taxis schedule, or arranges an empty taxi. T-Share [6] does consider dynamic scenarios, but its scheduling is rather simplistic: for each new request, it assigns the taxi with the minimum detour distance, without considering the experience of the passengers on the chosen taxi. Once a sharing schedule is made, it remains unchanged even though new requests may prefer a different schedule.

Beside the up-mentioned requirements for a practical taxi-sharing serve, we propose to considering the QoS of passengers. And for the first time proposed to dynamically adapting taxis' schedule as new requests arrive.

B. Carpooling

Carpooling is also known as the Dial-a-Ride Problem (DARP), where requests are static, i.e., routes and time schedules are computed in advance with the time constraints and capacity constraints satisfied. In theory, flourish of researches have been conducted on DARP [18, 13]. Since the DARP is NP-hard [18], only small instances can be solved optimally [13]. Large instances are usually solved by a two-step algorithm [19]. Specifically, the algorithm first clusters requests to be served by a single vehicle, and then routes each vehicle individually. Taxi-sharing is more challenging, since both passengers' requests and positions of taxis are highly dynamic and difficult to predict, yet we need to serve large-scale real-time requests quickly.

Considerably less research focuses on the dynamic DARP problem, where customer requests are generated on the fly [20, 21]. These approaches do not consider the QoS of passengers and the profit of drivers as a whole.

C. Taxi Serving Systems

Due to the ubiquitous deployment of GPS devices, the past two decades have witnessed extensive taxi serving systems. First, several systems are proposed for the benefit of passengers or drivers, based on the historical taxi traces. Yuan et al. [22] propose to suggest parking places for taxi drivers

where they can find passengers quickly and maximize the profit of the next ride. Zheng et al. [23] suggest waiting locations for passengers where they can find a taxi more easily. While these systems try to serve either drivers or passengers, our system aims to achieve mutual benefit. Second, taxi dispatching services [24] usually match the nearest vacant taxi for a request, which can be easily retrieved by indexing the moving objects. In our case, we need to satisfy both the time constraints and capacity constraints. This complication introduces new challenges.

VII. Conclusion

In this study, we propose a QoS-Aware taxi-sharing system, QA-Share. We formulate the optimization problem using integer linear programming, and derive the optimal solution under a small system scale. When the number of requests and taxis becomes large, we devise a heuristic algorithm that has a much faster execution time. We evaluate our approach with real-world dataset. We will incorporate a more accurate travel time estimation algorithm to improve the scheduling and calculation of taxi fare, in the future.

VIII. Acknowledgements

The authors thank the anonymous reviewers for their valuable feedbacks. This research is supported in part by NSFC under Grant No. 61472219, 61472218, 61402338, and NSFC Distinguished Young Scholars Program under Grant No. 61125202.

References

[1] N. Taxi and L. Commission, "Taxi of tomorrow survey," 2011.

[2] S. Daily. (2014) Taxi shift change a headache for locals. http://www.china.org.cn/english/travel/158628.htm. [Online; accessed 13-Dec-2014].

[3] C.-C. Tao and C.-Y. Chen, "Heuristic algorithms for the dynamic taxipooling problem based on intelligent transportation system technologies," in *Proceedings of IEEE FSKD*, 2007.

[4] P.-Y. Chen, J.-W. Liu, and W.-T. Chen, "A fuel-saving and pollution-reducing dynamic taxi-sharing protocol in vanets," in *Proceedings of IEEE VTC*, 2010.

[5] D. Zhang, Y. Li, F. Zhang, M. Lu, Y. Liu, and T. He, "coride: carpool service with a win-win fare model for large-scale taxicab networks," in *Proceedings of ACM SenSys*, 2013.

[6] S. Ma, Y. Zheng, and O. Wolfson, "T-share: A large-scale dynamic taxi ridesharing service," in *Proceedings of IEEE ICDE*, 2013.

[7] Y. Wang, Y. Zheng, and Y. Xue, "Travel time estimation of a path using sparse trajectories," in *Proceedings of ACM SIGKDD*, 2014.

[8] CNTV-English, "Beijing promotes taxi sharing to ease traffic jams," http://www.nihao-salam.com/news-detail.php?id=MTUxMQ==, 2012, [Online; accessed 30-July-2014].

[9] X. Liu, J. Biagioni, J. Eriksson, Y. Wang, G. Forman, and Y. Zhu, "Mining large-scale, sparse gps traces for map inference: comparison of approaches," in *Proceedings of ACM SIGKDD*, 2012.

[10] J. Zheng and L. M. Ni, "Time-dependent trajectory regression on road networks via multi-task learning," in *Proceedings of AAAI*, 2013.

[11] T. Idé and M. Sugiyama, "Trajectory regression on road networks," in *Proceedings of AAAI*, 2011.

[12] Z. He, J. Cao, and T. Li, "Mice: A real-time traffic estimation based vehicular path planning solution using vanets," in *Proceedings of IEEE ICCVE*, 2012.

[13] J.-F. Cordeau, "A branch-and-cut algorithm for the dial-a-ride problem," *Operations Research*, vol. 54, no. 3, pp. 573–586, 2006.

[14] J. E. Hopcroft and R. M. Karp, "An nˆ5/2 algorithm for maximum matchings in bipartite graphs," *SIAM Journal on computing*, vol. 2, no. 4, pp. 225–231, 1973.

[15] W. Wang, B. Li, and B. Liang, "Dominant resource fairness in cloud computing systems with heterogeneous servers," in *Proceedings of IEEE INFOCOM*, 2014.

[16] E. Blanton, S. Fahmy, and S. Banerjee, "Resource management in an active measurement service," in *Proceedings of the IEEE GIS*, 2008.

[17] S. C. Lee, J. W. Jiang, J. C. Lui, and D.-M. Chiu, "Interplay of isps: Distributed resource allocation and revenue maximization," in *Proceedings of IEEE ICDCS*, 2006.

[18] P. Healy and R. Moll, "A new extension of local search applied to the dial-a-ride problem," *European Journal of Operational Research*, vol. 83, no. 1, pp. 83–104, 1995.

[19] M. Aldaihani and M. M. Dessouky, "Hybrid scheduling methods for paratransit operations," *Computers & Industrial Engineering*, vol. 45, no. 1, pp. 75–96, 2003.

[20] Y. Lin, W. Li, F. Qiu, and H. Xu, "Research on optimization of vehicle routing problem for ride-sharing taxi," *Procedia-Social and Behavioral Sciences*, vol. 43, no. 0, pp. 494–502, 2012.

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

[22] J. Yuan, Y. Zheng, L. Zhang, X. Xie, and G. Sun, "Where to find my next passenger," in *Proceedings of ACM UbiComp*, 2011, pp. 109–118.

[23] X. Zheng, X. Liang, and K. Xu, "Where to wait for a taxi?" in *Proceedings of ACM SIGKDD*, 2012.

[24] J.-L. Lu, M.-Y. Yeh, Y.-C. Hsu, S.-N. Yang, C.-H. Gan, and M.-S. Chen, "Operating electric taxi fleets: A new dispatching strategy with charging plans," in *Proceedings of IEEE IEVC*, 2012.