An Efficient Dynamic Ridesharing Algorithm

Jianren Gao, Yuxin Wang, Haoyang Tang, Zhao Yin, Lei Ni, and Yanming Shen
School of Computer Science and Technology
Dalian University of Technology
Dalian, China

***Abstract*—In this paper, we propose a scheduling algorithm for ridesharing. It efficiently serves real-time requests sent by passengers and generates ride sharing schedules that try to meet users' demands. In our method, we first propose a taxi searching algorithm using binary search strategy to quickly and efficiently retrieve candidate taxis that are likely to satisfy a trip request. A scheduling algorithm is then proposed. It checks each candidate taxi and insert the request trip into the schedule of the taxi which satisfies the request with the maximum average satisfaction. We evaluated our system using a large scale taxi dataset containing 101,952 trips in Beijing Chaoyang district. Results showed that our approach serves 40% additional taxi users while saving 30% travel distance compared with no-ridesharing.**

Keywords—ridesharing, searching algorithm, scheduling algorithm

I. INTRODUCTION

With the explosive increase of car ownerships, it leads to severe traffic jams and pollution [1-2]. An efficient vehicle scheduling algorithm can improve the efficiency of transportation system, resulting in less congestions, shorter commute time, and less energy consumption.

Ridesharing is a promising approach. With the same number of passengers, ridesharing can carry more passengers and reduce the amount of vehicles in the road network, and therefore alleviate traffic congestions. At the same time, ridesharing can reduce the total travel distance of vehicles and reduce fuel consumption, and therefore reduce environmental pollution.

However, it is challenging to design an efficient ridesharing algorithm. Compared with no-ridesharing schedule, ridesharing schedule has more complex scheduling strategy. Since both passengers and taxis are highly dynamic, ridesharing is a process of dynamic scheduling optimization. In order to respond to a new request, one may have to reshuffle the predefined schedule and the reshuffled one has to be a valid schedule. Also, as trip requests arrive in real-time, we also need to generate a schedule quickly.

To address these challenges, in this paper, we first propose a taxi searching algorithm using binary search strategy to quickly and efficiently retrieve candidate taxis that are likely to satisfy a trip request, which can reduce the amount of calculations. A scheduling algorithm is then proposed. It checks each candidate taxi and insert the request trip into the schedule of the taxi which satisfies the request with the maximum average satisfaction. In our schedule, we don't consider reshuffle users' requests. Since if we reshuffle the predefined schedule when a new request arrives, passengers will be exchanged between different taxis. This cannot immediately determine which taxi to provide services for passengers, and therefore it is difficult to accurately respond to the user. Also note that in practice, it is extremely rare that the optimal schedule requires the schedule reordering [3].

The rest of this paper is organized as follows. In Section Ⅱ, we formally define the dynamic taxi ridesharing schedule problem. Section Ⅲ introduces the binary search algorithm based on time and the scheduling algorithm. We present the evaluation in Section Ⅳ and summarize the related work in Section Ⅴ. Conclusion is given in Section Ⅵ.

II. OVERVIEW

Definition 1 (Road Network): A road network $G = (V, E, W)$ consists of a vertex set V and an edge set E. Each edge $(v_1, v_2) \in E$ $(v_1, v_2 \in V)$ is associated with a weight $W(v_1, v_2)$ indicating the traveling cost along the edge (v_1, v_2), which can be a time or distance measure. Given two nodes o and d representing the starting point and the end point in the road network, the path p between them is a vertex sequence $(v_0, v_1, ..., v_n)$, where (v_i, v_{i+1}) is an edge in E, $v_0 = o$, and $v_n = d$.

Definition 2 (Trip Request): A trip request tr is denoted as $tr = (t, o, d, wp, wd, det, r, cnt)$. Among them, t indicates when this request is submitted, o is the pickup point, d is the destination point, a time window wp defines the time interval when the passenger needs to be picked up at the pickup point, wd and det indicate the detour time/distance that passenger can tolerate, r indicates the ratio for time window in this request, and cnt indicates the passenger count in this request.

Note that usually, detour distance is not equivalent to detour time. Therefore, in our trip request, we define wd and det respectively to make a finer scheduling decision. The detour distance det can be obtained using the shortest path, and the detour time wd can be obtained by comparing with the no-ride-sharing case.

978-1-5386-0958-3/17 $31.00 © 2017 IEEE
DOI 10.1109/CIT.2017.33

In order to facilitate the subsequent descriptions, we define the earliest and the latest time to pick up at o as $wp.e$ and $wp.l$, where

$$wp.e = t$$

$$wp.l = wp.e + wp\ .$$

Similarly, we define the earliest and latest time to drop off at d as $wd.e$ and $wd.l$, where

$$wd.e = t + min(t_{od})$$

$$wd.l = wd.e + wd\ .$$

Definition 3 (Taxi Schedule): A taxi schedule $S = (v_1, v_2, ..., v_{2n})$ is a temporally ordered sequence of pickup and destination points of n trip requests $(tr_1, tr_2, tr_3, ..., tr_n)$, where $v_i \in V, i = 1, ..., n$.

Definition 4 (Valid Taxi Schedule): A valid taxi schedule $S = (v_1, v_2, ..., v_{2n})$ is a temporally ordered sequence of pickup and drop-off points of n trip requests $(tr_1, tr_2, tr_3, ..., tr_n)$ satisfying the following four conditions:

1. Orderness: For any trip tr_i, let $v_{i_1} = o_i$, $v_{i_2} = d_i$. Then, we must have $i_1 < i_2$ (the index represents the position of the points in the trip schedule S), i.e., the pickup point must happen before its destination point;
2. Waiting time constraint: For any trip tr_i, the schedule must not exceed the pickup time and drop off time constraint;
3. Detour distance constraint: For any trip tr_i, the schedule must not exceed the detour distance constraint;
4. Passenger number constraint: For any taxi, the current number of passengers must not exceed the taxi capacity.

In order to facilitate the inspection of the time window constraints given by users, we define slack time to check whether the delay due to an insertion violates the timely arrival at any subsequent point in the schedule.

Definition 5 (Slack Time): Slack time is defined on the scheduling node. At any node v_i in a schedule S, the slack time for v_i is defined as $ST_{v_i} = T_{lateV_i} - T_{arriveV_i}$, where T_{lateV_i} is the latest allowed arrival time to v_i, $T_{arriveV_i}$ is the actual arrival time to v_i in schedule S. Only $ST_{v_i} \geq 0$ can meet the time window constraints of v_i.

Definition 6 (Service Rate): The service rate is the fraction of the trip request that are satisfied. It is a crucial criterion measuring the effectiveness of the ridesharing system.

Definition7 (Satisfaction): The satisfaction is defined on each trip request. For any trip request tr in schedule S the satisfaction is defined as

$$Sa_{tr} = tr(r)\frac{ST_{tr(o)} + ST_{tr(d)}}{tr(wp) + tr(wd)} + (1 - tr(r))\frac{tr(det) - detour}{tr(det)},$$

where $detour$ is the difference between the trip distance in request tr in schedule S and the trip distance without ridesharing.

If $tr(r) > 0.5$, it indicates that a user prefers to save more time. If $tr(r) < 0.5$, it indicates that a user prefers to reduce detour distance.

Definition 8 (Average Satisfaction): The average satisfaction is defined on a valid taxi schedule. For n trip requests $(tr_1, tr_2, tr_3, ..., tr_n)$ in any valid taxi schedule $S_i = (v_{i_1}, v_{i_2}, ..., v_{i_{2n}})$ the average satisfaction is defined as

$$AS_{S_i} = \frac{1}{n}\sum_{j=1}^{n} Sa_{tr_j}\ .$$

Let $Taxi = \{taxi_1, taxi_2, ..., taxi_n\}$ represent the set of taxis in the road network G. For any $taxi \in Taxi$, $taxi.location$ denotes its current position, $taxi.sq = (s_1, s_2, ..., s_{2m})$ means the scheduling queue for m requests $TR = (tr_1, tr_2, tr_3, ..., tr_m)$, $taxi.cnt_s$ means the number of passengers at s, $taxi.capacity$ is the capacity of the taxi. Let $dis_{s_i s_j}$ and $dis_{s_i \to s_j}$ denote the distance from s_i to s_j, without/with ridesharing respectively.

In this paper, the **Dynamic Taxi Ridesharing Problem** is defined as follows: given a set of taxis $Taxi$ on the road network G and a new incoming request tr_{new}, find the taxi $taxi \in Taxi$ that can get the maximum average satisfaction, as shown in (1).

$$\max_{taxi \in Taxi} \left(AS_{taxi.sq} \right) \tag{1}$$

subject to:

$$ST_s \geq 0, \forall s \in taxi.sq \tag{2}$$

$$dis_{tr(o) \to tr(d)} - dis_{tr(o)\, tr(d)} \leq tr(det), \forall tr \in TR \tag{3}$$

$$taxi.cnt_s \leq taxi.capacity, \forall s \in taxi.sq \tag{4}$$

Constraint (2) ensures the waiting time and detour time limit, constraint (3) ensures the detour distance limit and constraint (4) ensures the taxi capacity limit.

Note that when a new request is inserted into the scheduling queue, it will usually reduce the satisfaction of the passengers after this request. Therefore, if we maximize the current request satisfaction, the satisfactions of the requests after it will be reduced. That is why we use the average satisfaction in our formulation.

In our formulation, we try to maximize the average satisfaction for each taxi, and it does not guarantee that the average satisfaction of all taxis for all queries is maximized. The reason is that the problem of maximizing the average satisfaction of all taxis for the whole trip requests is NP-complete. We can prove this as follows. Let AS be the average total satisfaction of all taxis for the whole trip requests. Average Total Satisfaction Optimization Taxi Ridesharing Problem can be converted to the problem of minimizing the

total $1/AS$ of all taxis for the whole trip requests, which has already been proved to be NP-complete [6].

III. SEARCHING AND SCHEDULING ALGORITHM

Note that the number of candidate taxis may be very large, even thousands of taxis. If we check the constraint for each candidate taxi, it will lead to a high computation load and increase the response time. To address this issue, our approach is performed in two steps, searching and scheduling. In the first stage, we reduce the size of the candidate set by only looking at the time constraint. In the second stage, we check all constraints, and choose the taxi which achieves the maximum for Equation (1).

A. The binary search algorithm based on time

In order to guarantee service rate and reduce processing time, we propose a binary search strategy based on time. In this strategy, we set up two parameters, the minimum number of search for taxis and the maximum number of search for taxis. With a small candidate set, at the second stage, we may fail to find a valid taxi. A large candidate set cannot help reduce the computation. Therefore, we need to choose a suitable size of candidate set. With these two parameters, we can flexibly control the taxi count from the set of taxis that can provide services. Our algorithm works similar to binary search strategy, by adjusting the time limit, and it finally reaches the vehicle number limit. Therefore, it achieves a trade-off between realtime processing and maximizing the average satisfaction. Algorithm 1 shows the binary search algorithm based on time.

Algorithm 1: The binary search algorithm based on time

```
  Input: Trip request tr, all taxi status V_all, the
  minimum number of search for taxi min_taxi_cnt, the
  maximum number of search for taxi max_taxi_cnt.
  Output: Return a taxi candidate set.
1   taxi_candidate_set • the taxi candidate set,
  initialized to be ∅
2 for taxi status V in V_all do
3    if V satisfied tr.wp.l and tr.wd.l constraint then
4       add V to the taxi_candidate_set
5    end if
6 end for
7 size • the size of the taxi_candidate_set
8 while (size < min_taxi_cnt || size >= max_taxi_cnt)
&& tr.wp.e <= tr.wp.l && tr.wd.e <=  tr.wd.l do
9    o_mid • (tr.wp.e + tr.wp.l)/2
10   d_mid • (tr.wd.e + tr.wd.l)/2
11   taxi_candidate_set • ∅
12   for taxi status V in V_all do
13      if V arrived at tr.o and tr.d respectively meet
  the time limit o_mid and d_mid then
14         add V to the taxi_candidate_set
15      end if
16   end for
17   size • the size of the taxi_candidate_set
18   if size >= max_taxi_cnt then
19      tr.wp.l • reduce o_mid, e.g., one second
20      tr.wd.l • reduce d_mid, e.g., one second
21   else if size < min_taxi_cnt then
22      tr.wp.e • increase o_mid, e.g., one second
23      tr.wd.e • increase d_mid, e.g., one second
24   end if
25 end while
26 return taxi_candidate_set
```

Let m be the maximum time window (in seconds) and n be the number of taxis, and the time complexity of the algorithms can be expressed as $O(n \log m)$. We prove this statement as follows. Let $T(m,n)$ indicate the time complexity of the algorithm, then we have

$$\begin{aligned} T(m,n) &= T(m/2,n)+O(n) \\ T(m,n) &= T(m/4,n)+2O(n) \\ &\cdots \\ T(m,n) &= O(n \log m) \end{aligned}$$

Therefore, the time complexity of the algorithms can be expressed as $O(n \log m)$. □

Since usually m is much less than n, therefore $O(n \log m) \approx O(n)$. This means that the time complexity of the binary search algorithm is approximately linear to the number of taxis.

B. Scheduling algorithm

Compared with the detour distance and taxi capacity constraints, time window constraint is relatively easy to compute. Therefore, to quickly obtain a rough candidate taxi set, the binary search algorithm only considers the time constraint of the trip, and does not take into account the detour distance constraint and the taxi capacity constraint. From the set of taxis generated by the binary search, we need to choose an optimal taxi to serve the passenger. We use Algorithm 2 as an insertion feasibility check. In our design, we get the taxi from the search result set which can maximize the average satisfaction. If the algorithm returns the taxi candidate set that cannot provide service to a passenger, we will go back and search for the rest of other taxis which also satisfy the user time window limit.

Algorithm 2: Insertion feasibility check

```
  Input: Trip request tr, taxi status V, insertion
position i for tr.o, insertion position j for tr.d.
  Output: Return average satisfaction if the request can
be inserted; otherwise return 0.
1 for any point s in V.sq do /* V.sq is the scheduling
queue of V */
2    if V.cnt_s > V.capacity || ST_s < 0 then /* V.cnt_s
means the number of passengers at  s, ST_s means the
slack time at s */
3       return 0
4    end if
5 end for
6 for any trip tp in V.sq do
7    if the detour distance for tp incurred by the
  insertion of tr.o or tr.d beyond the tp.det limit then
8       return 0
9    end if
10 end for
11 avg_satisfaction • compute average satisfaction of
  inserting tr.o and tr.d into V.schedule
12 return avg_satisfaction
```

IV. EVALUATION

A. Experiment Setting

We evaluate the system using a large scale taxi dataset, which contains 101,952 trips in Beijing Chaoyang district over one day generated by a taxi request simulator [5]. The simulator can generate requests for taxicabs on different road segments, using a GPS trajectory dataset generated by over 33,000 taxis during a period of 3 months in Beijing, China. Each query consists of an origin, destination, and a timestamp. Note that our trip request tr is with this format

$tr = (t, o, d, wp, wd, det, r, cnt)$, where t is the timestamp, o is the origin, d is the destination, wp and wd can be set as the default values of 5min, 10min or 15min , det is defined as trip distance, r is 0.5, cnt is 1. For sake of simplicity, we assume that all passengers have similar deadlines. Fig. 1 shows the query distribution over time.

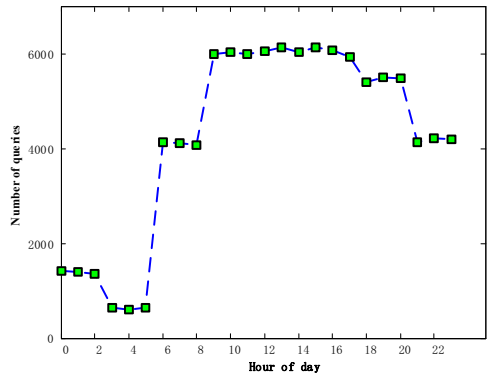

Fig. 1 Query distribution

Fig. 2 shows the travel time distribution of trip requests. We can see that a lot of trips are short, which facilitates the ridesharing.

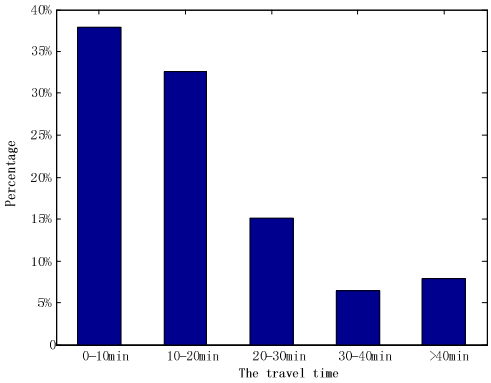

Fig. 2 Trip travel time statistics

The Beijing road network is represented by an undirected and weighted graph containing 346,299 vertices and 376,968 edges. The road network is stored in MongoDB. In this paper, we calculate the distance between two points in road network using Dijkstra algorithm. Since travel time estimation itself is not the focus of this paper, we assume a constant speed. Specifically, according to the traffic in Beijing, we set the speed to 22.1 km/hr.

For large scale ride sharing, the shortest path algorithm is called very frequently. In order to improve the computing speed, we divide the map into grids. When we calculate the distance between two points on the road network, we use the grid distance as an approximation. In our experiments, the grid size we use is 700 m. In order to further improve speed we also use cache to store the shortest paths. A new shortest path is calculated when cache miss happens. The cache is indexed only by origin and destination points in a path computation call.

The simulation framework is implemented in Java. We run experiments on a laptop with an Intel(R) Core(TM) i5-24330M (2.40GHz) processor, and the RAM is 4GB. This simulation implementation is multi-threaded, and memory usage is limited to two gigabytes.

B. *Experiment Results*

Let SR denote the service rate (fraction of requests serviced in the ridesharing), AS denote the average satisfaction, and CRP denote the computation reduction proportion. We first need to determine the optimal range limit for our binary search algorithm. As Fig. 3 shows, SR, AS and CRP have the same change trend over the range limit for different time windows. The greater the time window, the higher the service rate, and the lower the average satisfaction. But both the service rate and the average satisfaction can be kept at a high level. The calculation reduced proportion is a parabolic curve, which is consistent with our algorithm.

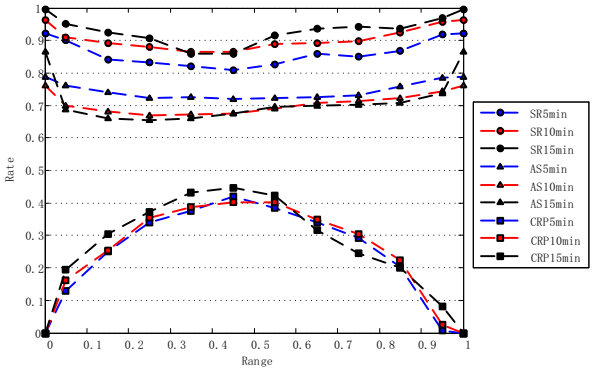

Fig. 3 The SR, AS and CRP experimental results

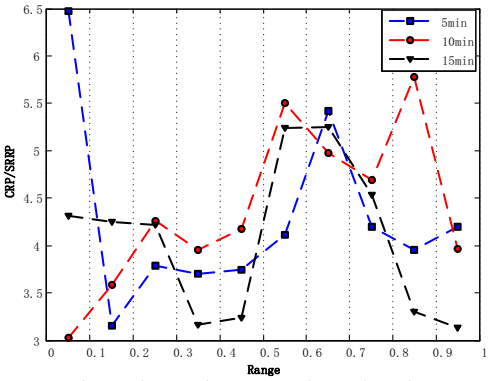

Fig. 4 The CRP/SRRP experimental results

Let SRRP denote the service rate reduced proportion. Fig. 4 shows how CRP/SRRP changes over the range limit for different time windows. Note that a bigger CRP is better, and a smaller SRRP is better, and therefore a bigger CRP/SRRP is better. As can be seen from the figure, the range limit of 0.6 to 0.7 can achieve the best effect. Therefore in the subsequent experiments, we choose the range of 0.6 to 0.7 as the range limit of our binary search algorithm. It should be noted that the range here refers to the percentage of taxis that meet the limited time window specified by the trip request.

Next, we compare our algorithm with the case of no-ridesharing. As shown in TABLE □, compared with no-ridesharing, our proposed carpool scheduling method can reduce the driving distance by 30%. Fig. 5 and Fig. 6 show the service rate of our algorithm and no-ridesharing respectively. As taxi requests increase in peak hours, service rate will fall. Without ridesharing, the service rate is reduced to about 40%. But with our ridesharing algorithm, we can still maintain a relatively high service rate.

TABLE I. COMPARISON OF RIDESHARING AND NO-RIDESHARING

Carpooling	wp wd	Taxi Count	All Service Trip Distance	Driving Distance	The Distance Reduced Proportion
yes	5min	852	556042.3 380km	379241.3 130km	0.3180
no	5min	852	302923.7 790km	343445.7 880km	-
yes	10min	852	575305.1 900km	404607.4 700km	0.2967
no	10min	852	329120.0 850km	375030.4 830km	-
yes	15min	852	609461.7 660km	405753.5 160km	0.3342
no	15min	852	329751.5 240km	380837.6 260km	-

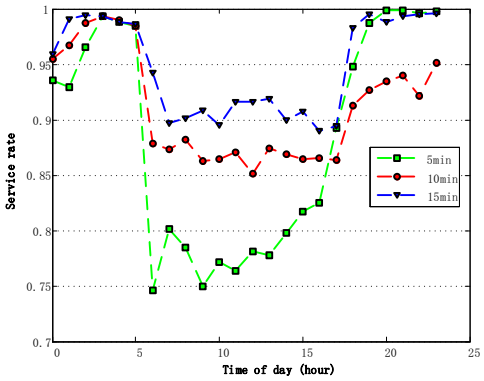

Fig. 5 Service rate of our algorithm

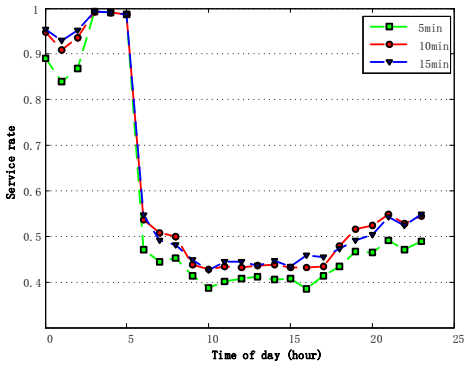

Fig. 6 Service rate of no-ridesharing

V. RELATED WORK

Sharing taxi trips is a possible way of reducing the negative impact of taxi services on cities [4]. The main challenge in dynamic ride sharing is to determine how to handle trip requests as they enter into the system in real-time. [5] formalized and studied the slugging problem which is a particular ridesharing form and proposed a quadratic algorithm to solve it optimally. [6] formally defined the dynamic ridesharing problem and propose a large-scale taxi ridesharing service based on the grid and spatial-temporal indexing. [7] proposed a distributed cloud computing strategies to make ridesharing scalable. [8] introduced the problem of large scale real-time ridesharing with service guarantee on road networks and proposed a kinetic tree algorithm which is better suited to efficient scheduling of dynamic requests and adjust routes on-the-fly. [9] proposed the Hybrid-simulated annealing (HSA) algorithm to dynamically assign passenger requests. [10] presented a classification to understand the key aspects of existing ridesharing systems. Shen et.al [11] formally address the problem of dynamic ridesharing and introduce the solution framework of filter and refine, under which they summarize existing state-of-the-art works. In [12], the authors proposed a novel system, MARS (Multi-Agent Ridesharing System), which formulates travel time estimation and enhances the efficiency of taxi searching through a decremented search approach. [13, 14] give an overview of the dynamic taxi-sharing system, present a research framework to explore the internal and external factors affecting the customer's acceptance with taxi-sharing services by using multivariate analysis methods, describe the empirical results of the field trial and provide valuable implications to better taxi-sharing services in the future.

Research efforts on analysis of taxi ride sharing problem have been undertaken to consider different constraints, e.g., vehicle capacity constraint [14], waiting time constraint [8], money constraint [15], and combination of these constraints [3, 6, 16]. Our work also considers all these constraints except money constraint, but the detour distances constraint can be considered as equivalent to money constraint.

VI. CONCLUSION

This paper proposed a dynamic taxi ridesharing algorithm. First, our proposed binary search algorithm based on time can significantly reduce the amount of calculations, at the cost of slightly reducing service rate and average satisfaction. Then, our ridesharing service can effectively reduce the taxi driving distance. We evaluated our system based on a large scale taxi dataset. The experimental results show that our system is effective and efficient in serving dynamic trip requests. For example, our proposed algorithm serves 40% additional taxi users while saving 30% travel distance compared with no-ridesharing and the average satisfaction is increased by almost 5%. In the future, we will adopt advanced travel time estimation techniques to improve the taxi travel time estimation accuracy. We will also consider refining the ridesharing problem by considering other constraints, such as money constraints, gender preference, etc.

Acknowledgment

This work is supported in part by the Fundamental Research Funds for the Central Universities (DUT17ZD303), and also in part by NSFC under grant No. 11372067 and 61173160.

REFERENCES

[1] Morency C. The ambivalence of ridesharing. Transportation, 2007, 34(2): 239-253.

[2] Chan, N.D., Shaheen, S.A.: Ridesharing in North America: past, present, and future. Transp. Rev. 32, 93–112 (2012).

[3] Ma S, Zheng Y, Wolfson O. Real-Time City-Scale Taxi Ridesharing. IEEE Transactions on Knowledge & Data Engineering, 2015, 27(7):1782-1795.

[4] Santi P, Resta G, Szell M, et al. Quantifying the benefits of vehicle pooling with shareability networks. Proceedings of the National Academy of Sciences, 2014, 111(37): 13290-13294.

[5] Ma S, Wolfson O. Analysis and evaluation of the slugging form of ridesharing. Proceedings of the 21st ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems. ACM, 2013: 64-73.

[6] Ma S, Zheng Y, Wolfson O. T-share: A large-scale dynamic taxi ridesharing service. IEEE 29th International Conference on. Data Engineering (ICDE), 2013: 410-421.

[7] Dimitrijević D, Nedić N, Dimitrieski V. Real-time ridesharing and ride-sharing: Position paper on design concepts, distribution and cloud computing strategies. Computer Science and Information Systems (FedCSIS), 2013 Federated Conference on. IEEE, 2013: 781-786.

[8] Huang Y, Jin R, Bastani F, et al. Large Scale Real-time Ridesharing with Service Guarantee on Road Networks. Proceedings of the VLDB Endowment, 2013, 7(14).

[9] Jung J, Jayakrishnan R, Park J Y. Dynamic Shared-Taxi Dispatch Algorithm with Hybrid-Simulated Annealing. Computer-Aided Civil and Infrastructure Engineering, 2016, 31(4):275-291.

[10] Furuhata M, Dessouky M, Ordóñez F, et al. Ridesharing: The state-of-the-art and future directions. Transportation Research Part B: Methodological, 2013, 57: 28-46.

[11] Shen B, Huang Y, Zhao Y. Dynamic ridesharing. Sigspatial Special, 2016, 7(3):3-10.

[12] Shemshadi A, Sheng Q Z, Zhang W E. A Decremental Search Approach for Large Scale Dynamic Ridesharing/ Web Information Systems Engineering – WISE 2014. Springer International Publishing, 2014:202-217.

[13] Tao C, Wu C. Behavioral responses to dynamic ridesharing services - The case of taxi-sharing project in Taipei. IEEE International Conference on Service Operations and Logistics, and Informatics, 2008. IEEE/SOLI. 2008.

[14] Tao C C. Dynamic Taxi-sharing Service Using Intelligent Transportation System Technologies. International Conference on Wireless Communications, NETWORKING and Mobile Computing. 2007:3209 - 3212.

[15] Santos D O, Xavier E C. Taxi and Ride Sharing: A Dynamic Dial-a-Ride Problem with Money as an Incentive. Expert Systems with Applications, 2015, 42(19):6728–6737.

[16] Duan X, Jin C, Wang X, et al. Real-Time Personalized Taxi-Sharing. Database Systems for Advanced Applications. Springer International Publishing, 2016.