Real-Time Distributed Taxi Ride Sharing

Conference Paper · November 2018

DOI: 10.1109/ITSC.2018.8569315

4 authors, including:

Vaskar Raychoudhury
Miami University
66 PUBLICATIONS 650 CITATIONS
SEE PROFILE

Divya Saxena
Indian Institute of Technology Roorkee
25 PUBLICATIONS 296 CITATIONS
SEE PROFILE

Ajay D Kshemkalyani
University of Illinois at Chicago
145 PUBLICATIONS 2,706 CITATIONS
SEE PROFILE

2018 21st International Conference on Intelligent Transportation Systems (ITSC)
Maui, Hawaii, USA, November 4-7, 2018

Real-Time Distributed Taxi Ride Sharing

Kanika Bathla, Dept. of CSE, IIT Roorkee, Roorkee, India, kanikabathla16@gmail.com

Vaskar Raychoudhury, Dept. of CSE, Miami University, Oxford, OH, USA, vaskar@ieee.org

Divya Saxena, Dept. of Computing, The Hong Kong PolyU, Kowloon, Hong Kong, divya.saxena.2015@ieee.org

Ajay D Kshemkalyani, Dept. of CSE, Univ. of Illinois at Chicago, Chicago, IL, USA, ajay@uic.edu

***Abstract*— Taxicabs play an important role in urban public transportation. Analyzing taxi traffic of Shanghai, San Francisco, and New York City, we have found that the short trips within city are mostly of commuters during office hours and span a specific city area. Now, if the large number of commuters are ready to share their rides, that will have a huge impact on the 'super-commute' problem faced in various cities of USA and around the world. While ride-sharing can increase taxi occupancy and profit for drivers and savings for passengers, it reduces the overall on-road traffic and thereby the average commute time and carbon foot-print. While centralized ride-sharing services, like car-pooling, can address the problem to some extent, they lack scalability and power to dynamically adapt the taxi schedule for best results. In this paper, we propose a four-way model for the ride-sharing problem and develop a novel distributed taxi ride sharing (*TRS*) algorithm to address dynamic scheduling of ride sharing requests. Our algorithm shows the overall reduction in total distance travelled by taxis as a result of ride sharing. Empirical results using large scale taxi GPS traces from Shanghai, China show that TRS algorithm can grossly outperform a Taxi Distance Minimization (TDM) algorithm. TRS accommodates 33% higher ride share among passengers while dealing with 44,241 requests handled by 4,000 taxis on a single day in Shanghai.**

Keywords—Taxi ride sharing; Smart Transportation; Distributed Coordination; Data Analysis; GPS traces.

I. Introduction

Transportation in big cities is often a nightmare for commuters mainly during peak hours of the working days. In this paper, we focus on taxis as a convenient and comfortable medium of public transport. Since, number of taxis often falls short of demand, during peak hours, passengers may have to queue up for a long time to get a taxi. While increasing number of taxis in a city will lead to more traffic congestion and growing carbon footprint, ride sharing can address those problems, given the willingness of passengers to share rides.

Fig. 1(a) and (b) shows the heat map of the taxi pickup and drop-off locations, respectively for a particular day (February 20, 2007) in Shanghai, China. The regions of higher intensity are shown as red and the regions of lower intensity are shown as green. Close observation shows that the pickup and drop-off locations are spread across a particular area of the city and are not widely distributed. From this, we can conclude that commute of people usually shares common routes. That means, it is possible for two or more passengers to share a complete ride or a part of it as long they are spatially and temporally co-located for the whole or part of the trip. Efficient ride sharing systems can decrease passenger's waiting time, travel cost (using cost sharing among riders) and carbon footprint while increasing average taxi occupancy and thereby driver's profit.

Ride sharing can be either static (pre-coordinated offline between a number of passengers and does not change afterwards, like Dial-a-Ride [1]-[6] or carpooling [7][8]) or dynamic (real-time and online). Static ride sharing solutions collect all the requests and apply various heuristics or optimization based strategies to achieve the best results. Dynamic Ride Sharing, on the other hand, is non-trivial as it requires a moving taxi to continuously adapt its schedule in real time in order to accommodate *eligible* (which are possible to serve) incoming requests. An incoming request is considered eligible by a taxi, only if the associated pick up and drop off events are not in conflict with the already accepted events in the current schedule of the taxi.

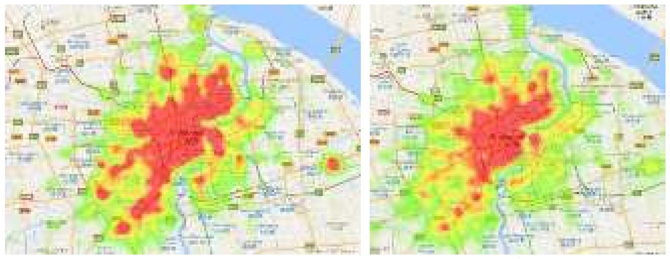

(a) (b)

Fig. 1. Heat map for (a) pickup and (b) drop-off locations

Existing solutions for dynamic ride sharing are either centralized or distributed. Centralized solutions require all passengers and taxi drivers to communicate to a central server, which does a back-end match making and often becomes a bottleneck due to the lack of scalability. Distributed Real Time Taxi Ride Sharing Solutions operate by independent localized co-ordinations between passengers and Taxis using message passing. However, to the best of our knowledge, there is no distributed taxi ride sharing algorithm with extensive experimentation using real taxi GPS traces.

In this paper, we have proposed four different models for distributed ride sharing and developed a generic algorithm. We assume synchronous wireless messaging system with real-time responses. In summary, we make the following contributions in this paper:

- Four different models for ride sharing have been proposed depending on the pickup and drop off locations of passengers interested in ridesharing. The models vary from most constrained to most generic in terms of flexibility of ride sharing.

- A novel distributed algorithm for ride sharing among numerous passengers has been proposed which works by localized communication between nearby passengers and taxi drivers. The algorithm accepts real-time ride requests and checks whether they can be accommodated into the existing taxi schedule with nominal finite delay to the other passengers.

- We have evaluated our proposed scheme using large scale GPS traces of 4,000 taxis in Shanghai. We have successfully adapted the dataset pertaining to individual taxi rides for testing our ride sharing algorithm and this step in itself was quite challenging. Our extensive experimentation reveals that ride sharing can effectively increase driver's profit and passenger's savings while reducing the total distance travelled (without ride-sharing).

II. Literature Survey

Ride sharing in public vehicles is an interesting problem which has received recent attention from multiple researchers. The existing literature in this area can be divided into *static* and *dynamic* ride sharing.

A. *Static Ride Sharing Solutions*

Static Ride Sharing are of different types - dial-a-ride, carpooling and slugging. Dial-a-Ride Problem (DARP) [1] is a NP-hard problem and existing papers try to intelligently group passengers for ride sharing so that the travel costs can be minimized on various routes. They either use integer programming based optimization [2] or adopt various heuristics to solve large static DARP instances [1][3][4][5][6]. Carpooling is a regular commute service of passengers like going to office from home or vice versa. It does not take into consideration the trip request which are generated in real time as discussed in [7][8]. Slugging is a typical form of ride sharing where passengers walk to the origin of the driver's trip, board at the departure time, debark at the driver's destination and then walk to their own destinations [9].

B. *Dynamic Ride Sharing: Centralized Approach*

Dynamic ride sharing problem considers that the requests from passengers are generated in real-time and on-the-fly. So, we need algorithms to match ride requests with the available vehicles in real-time while aiming to optimize the costs. Based on the solution approach, existing works on dynamic ride sharing can be either centralized or distributed.

Centralized schemes have been thoroughly surveyed in [10] with respect to the adopted optimization strategies that match drivers and riders in real-time. An opportunistic user interface to support centralized ride sharing planning has been designed in [11] while preserving location privacy. In [12], taxi searching, scheduling process is performed in cloud. In [13], central server is used for searching and scheduling. In [14], rides are considered to be "one to many" (one pick up but different drop off locations), and "many to one" (one drop off location but many pick up locations) and there is a central server. These approaches are not scalable.

Taxi searching is an important module in taxi ride sharing. In [12][15][16][17], the process of searching a taxi is facilitated by partitioning the road network into square grids. In [13][16][17], branch and bound and integer programming solutions are proposed although they are not dynamic. Authors in [18] propose to reduce the taxi search space using two constraints, such as seat availability and destination closeness but do not consider reducing passenger waiting time. In order to address the scalability issues of centralized systems, [19] has proposed to use Hadoop MapReduce and [20] uses cloud computing. In [21], the main focus is on matching of passengers with drivers with minimum matching latency.

Heuristic algorithms to solve the problem of taxi ride sharing has been proposed in [6][22][23][24][25]. In [25], passengers are transferred between multiple drivers to reduce the total miles travelled by the taxi. However, the system is not flexible for passengers because they have to switch between multiple drivers in between their journey. We have implemented this algorithm and compared it with the performance of our proposed algorithm (see *Section V(G)*).

In [26], commuters connect with each other through social network like Twitter and arrange rides with each other. However, the information about traffic and the constraints for passengers have not been taken into account and in [27], mobile phone Call Detail Records (CDRs), Twitter, and Foursquare data have been used for the trips between home and work location. In [28], close by requests are grouped together to utilize cab space efficiently. Parallel algorithms along with space partitioning techniques have been used to improve the scalability of the system.

C. *Dynamic Ride Sharing: Distributed Approach*

An algorithm for dynamic ride sharing through distributed coordination among vehicles and passengers has been proposed in [29], which first inserts a new request at the end of a taxi schedule and then calculates all the possible permutations of the routes which the taxi can follow. A purely distributed and very loosely coupled system for ride sharing has been proposed in [30]. However, this system is more conceptual and is not tested. In [31], based on the geometry matching, passengers are matched with vehicles considering the traffic information obtained through message exchanges.

Vehicle to vehicle (V2V) and Vehicle to Road (V2R) communications have been used in [32] to provide ride sharing services. Carpooling information collected by Road Side Unit (RSU) is broadcasted within its one hop communication range. Geometric method is used by vehicles to create travel path for carpooling service. In [33], the transportation network has been considered as an ad-hoc mobile geo-sensor network with different communication strategies for dissemination of ride sharing requests. In [34], an agent based concept is used for real time carpooling. Dijkstra algorithm has been used in distributed manner to find the shortest path for route calculation. In [35], the concept of detour has been explained. Two more interesting dynamic solutions are proposed in [36][37]. In summary, there is no fully functional distributed algorithm for taxi ride sharing.

III. Definitions, Variables and System Model

In this section, we present four different system models for ride sharing depending on the pickup and drop off locations of passengers interested in ridesharing. Before presenting our system models, we formally define some key terms and concepts used in our algorithm.

Event*:* Taxi considers each REQUEST (see Table II) from a passenger as a composition of two events, one *pickup (e_{pick})* and the corresponding *drop-off (e_{drop})*. Each event (*e*) has some attributes like *Id*, *location* (latitude and longitude), *time of occurrence* and a Boolean *flag* to distinguish between pickup (flag = 0) and drop-off (flag = 1) events.

Taxi Schedule / Event order schedule*:* Every taxi has two schedules - temporary and permanent and they consist of a sequence of events ordered by their timestamps of occurrence (not timestamp of *REQUEST* arrival). Each taxi inserts events from a *REQUEST* in its temporary schedule and checks for their *eligibility* to incorporate in the permanent schedule. Events in the temporary schedule are *eligible* iff they are not in conflict with events in the permanent schedule. Expired events from the permanent schedule are deleted during each update. Taxis operate only as per their permanent schedule.

A. *Variables and Data Structures*

In this section, we have listed the variables and data structures (Table I) as well as messages (Table II) used in the description and the pseudocode of our proposed algorithm.

TABLE I. VARIABLES AND DATA STRUCTURES USED IN ALGORITHM

Variables	Significance
P_{id}	Unique integer identifier of a Passenger *P*
S_p	Pick up location (latitude, longitude) of *P*
D_p	Drop o location (latitude, longitude) of P
t	Time at which P made a ride sharing request
MAX_WAIT	Maximum time (mins.) up to which *P* wait for pickup
duration	Time interval for which *P* travels in taxi. (journey time)
Δ	Maximum Time (in mins.) up to which the *duration* can be exceeded from the actual time of trip (slack time)
T_{id}	Unique integer identifier of a Taxi *T*
T_{loc}	Current location (latitude, longitude) of *T*
speed	Average speed of each taxi (40 km/hr.)
N	Capacity (in integer) of *T*, it is set as 4
N_{vac}	Number of available seats (in integer) in *T*
e	Pick up (e_{pick}) or drop-o (e_{drop}) event
Q_p	Permanent schedule at *T* for final storage of *e*
Q_t	Temporary schedule at *T* for temporary storage of *e*
pick_ index	The position where *e* is inserted in Q_t
$time_p$	Time (in minutes) estimated by *T* to reach S_p
$time_d$	Time (in mins.) estimated by *T* to reach D_p (considering current state of Q_t and road congestion)
C_{km}	Cost per km, we have assumed it to be Rs. 22/km
C_p	Estimated total cost (real value) for the ride of *P*.
time (a, b)	Time (in minutes) estimated by *T* to reach *b* from *a* where *a* and *b* are (latitude, longitude) pairs
dist (a, b)	Distance (in kms) estimated by *T* between *b* and *a* where *a* and *b* are (latitude, longitude) pairs
L	Sorted (w.r.to C_p) list of REPLY messages stored by *P*.
status	Boolean flag to indicate if taxi is selected (*status = true*) or rejected (*status = false*)
actual time	Time taken to travel directly from S_p to D_p

TABLE II. MESSAGES USED IN ALGORITHM

Messages	Significance
REQUEST (P_{id}, Sp, Dp, t)	Request for ride sharing sent by P
REPLY (T_{id}, $time_p$, $time_d$, C_p)	Reply from *T* to *P* with estimated pick up time, drop off time and cost for ride
CONFIRMATION (P_{id}, status)	Acceptance / Rejection status sent by *P* to *T*

B. *System Model and Assumptions*

Given a set of passengers $P = \{p_1, p_2,\ldots, p_n\}$ and a set of taxis T= $\{t_1, t_2,..., t_m\}$, the models (Table III) are as below:

- **Model 1**: For all $p_i \subseteq P$ and $p_i \leq N$, the pickup and drop off locations are same. This is the most constrained case and can be statically scheduled. There is only one pair of event in the event order schedule.
- **Model 2**: For all the users, $u \subseteq P$ and $u \leq N$, the pickup location is same (e.g., Airport, Rail Stn.) but they have different drop off locations. For *n* number of passengers, the total number of events in taxi's schedule is bounded by (*n+1*).
- **Model 3**: For all the users, $u \subseteq P$, the pickup locations are different but they have same drop off location (e.g., Airport, Stadium). For *n* passengers, total number of events is bounded by (*n+1*).
- **Model 4**: For all the users, $u \subseteq P$, the pickup and drop off locations are different. **This case is most generic** and the taxi starts after picking up the first passenger and decides on other passengers while on the move (dynamically). For *n* number of passengers, the event order schedule may contain maximum (*n* x 2) events.

TABLE III. MODELS OF RIDE SHARING

Pickup Location / Drop-off Location	Fixed	Different
Fixed	M 1/ static	M 3/ dynamic
Different	M 2/ static	M 4/ dynamic

In this paper, we have proposed a dynamic algorithm to solve the most generic model (Model 4) since the other models are more constrained cases of the M4. Both a taxi and a passenger check *eligibility* of a new *REQUEST* as per the following constraints:

- The speed of eligibility determination of a request is much faster than the taxi speed.
- Every passenger request is actually a ride sharing request for one seat at a time and taxis must commit as per their capacity: Empty => $N_{vac} = N$, Full => $N_{vac} = 0$, Otherwise => $0 < N_{vac} < N$.
- Waiting time of a passenger is upper-bounded by MAX_WAIT.
- The pickup event timestamp should always precede the drop-off event timestamp for the same *REQUEST*. (*precedence rule*).
- Promised pickup and drop-off times for committed passengers are strictly upper bounded by *Δ* (No Conflict rule).

IV. TAXI RIDE SHARING (TRS) ALGORITHM

Our algorithm (Fig. 2) starts when a passenger (P_i) broadcasts a *REQUEST* message to all the taxis within the wireless transmission range of 200 meters. A taxi on receiving a *REQUEST*, inserts the corresponding events in its temporary schedule (Q_t) and checks their eligibility (lines 7-9) using the *Schedule ()* procedure. If the events are eligible, the taxi sends

REPLY to *P* and waits for *CONFIRMATION* (line 10-11). *P* collects all the *REPLY* and sorts it w.r.to cost, available seats (objective is to maximize occupancy) and minimum time to destination and sends *CONFIRMATION* to the most suited taxi (line 3-6) with a *true* status and *false* to others. On receiving a *CONFIRMATION* with *true* status, the taxi updates the eligible events into the permanent schedule (Q_p), otherwise, the events are removed from Q_t (line 12-15).

Eligibility of a request is the possibility of the corresponding pick up and drop off events to be incorporated into the permanent schedule of a taxi and is evaluated using the *Schedule ()* function. For *Pickup Scheduling*, a taxi finds the appropriate value of the *pick index* to insert the pickup event of *P*. If the taxi is full, then pickup event can be inserted only after the first existing drop-off event in Q_t. Let insertion of pickup event (e_{pick}) is checked between events e_j and event e_k (see Fig. 3), where *j* and *k* are any integer values between 0 and size of schedule length, and $j<k$. Before inserting the e_{pick}, following constraints are checked which are self-explanatory.

```
Pseudo Code at Passenger's end
1.  L ← Ø
2.  Broadcast REQUEST message
3.  L.append (REPLY message) \\collect REPLY messages
4.  Sort L based on $C_p$ and then based on $N_{vac}$ and then based on $time_d$
5.  Select the taxi from L where wait time < MAX_WAIT and
    duration ≤ actual time+ Δ
6.  Send CONFIRMATION message to the selected taxi with status as
    true and with status as false to the rejected taxis
Pseudo Code at Taxi's end
7.  pick_ index ← Schedule ($Q_t$, P ,1, $S_p$)
8.  drop index ← Schedule ($Q_t$, P, pick_index+1, $D_p$)
9.  Insert pickup and drop-off in $Q_t$
10. Send REPLY message to P
11. Wait for CONFIRMATION message from P
12. if status=true then
13.         Insert pickup and drop-off in $Q_p$
14. else
15.         Remove pickup and drop-off event from $Q_t$
Schedule ($Q_t$, P, index, e.loc)
16. while index < $Q_t$.size() do
17.     time1←time($Q_t$[index-1].loc, $Q_t$[index].loc)
18.     time2←time($Q_t$[index-1].loc, e.loc)
19.     time3←time (e.loc, $Q_t$[index].loc)
20.     if time1 > time2 then
21.         if $Q_t$[index-1]. time + time2 + time3 ≤ $Q_t$[index].time then
22.             time← $Q_t$ [index-1]. time + time2
23.             return index
24.     index++
25. schedule_length← $Q_t$.size()-1
26. time2←time(Q[schedule_length]. loc, e.loc)
27. time ←$Q_t$[schedule_length].time + time2
28. return schedule_length+1
```

Fig. 2. Pseudocode of Taxi Ride Sharing (TRS) Algorithm

$$time\ 2(e_j,\ e_{pick}) < time\ 1(e_j,\ e_k) \qquad (1) \qquad [\text{line } 20]$$

$$time\ 2(e_j,\ e_{pick}) + time\ 3(e_{pick},\ e_k) \le e_k.time \qquad (2) \qquad [\text{line } 21]$$

If there is no suitable interval in the taxi schedule is found to insert the pickup event, it is inserted at the end of the taxi schedule (line 25-28). The drop-off scheduling works similar to the pick-up scheduling keeping in mind that the drop-off event of P can be inserted after pickup event of P (precedence rule). This is ensured by passing index as *pick_index+1*.

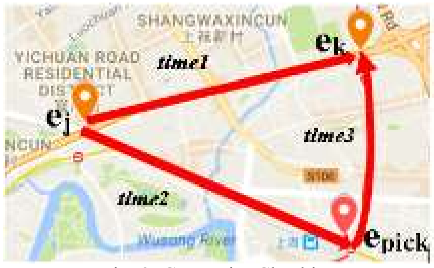


Fig. 3. Constraint Checking

V. Experiments and Results

In this section, we describe the performance of our proposed algorithms.

A. Dataset

The biggest difficulty of testing our ride-sharing algorithm is that there is no ride-sharing related data available. So, we use a large-scale taxi GPS traces of Shanghai, China [38] for testing our proposed algorithm. The Shaghai dataset consists of GPS traces of 4,000 taxis collected over a duration of more than two years. The taxis operates in the whole urban area of Shanghai, which covers an area of over 120 square kilometers and sends GPS signals with periods varying from 10 seconds to several minutes.

The dataset has attributes like *taxi id, latitude, longitude, speed, angle, date-time* and the Boolean *status* of the taxi (with / without passenger). We separated taxi-specific data for entire days and then sorted it, first date-wise and then pick-up time-wise (when the *status* variable changes to 1). In order to simulate the scenario of multiple requests for ride-sharing being submitted from the same or nearby regions, we have applied DBSCAN clustering on the real data set and assigned the requests that were within half *km* range of each other, to the same cluster. After finding the cluster for each request, requests were sorted based on cluster id so that requests from nearby regions can be grouped together. We have assumed that requests are submitted from the sorted list of requests to ensure that the requests are coming from places which are nearby. Since, we do not have the coordinates of path followed by a taxi, it is assumed to follow the discrete location coordinates of different events it serves and the taxi location is updated from the location of previous event to that of current event at the specific time of occurrence of those events.

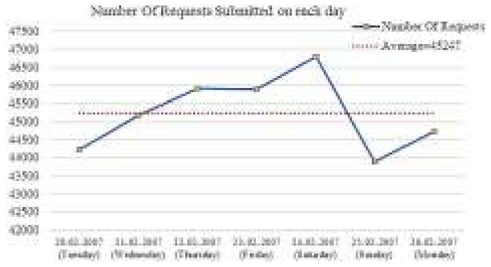


Fig. 4. Number of Requests submitted on each day

We have performed experiments using data from February 20-26, 2007. The total number of requests submitted on each of these days is shown in Fig. 4, which peaks on 24-02-2007 (Saturday). Fig. 5 analyzes hour-wise the data for 20-02-2007.

Broadcasting of REQUEST message is done within a range of 200 meters. We have calculated this distance using

Haversine formula and the distance and time between two locations is calculated at taxi's end using Google Map Direction API [31].

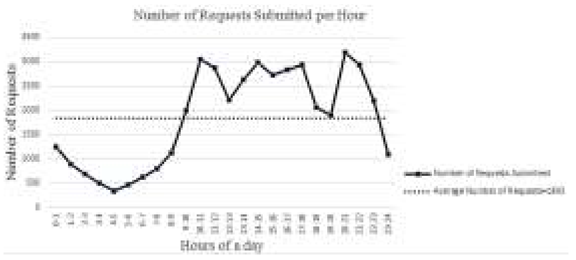


Fig. 5. Number of Requests submitted on 20-02-2007 per hour

B. Ride Sharing Cost Calculation

In a ride sharing, passengers may share some portion of distance with other peers in a taxi. So, the cost for the distance which is shared among 2 or more passengers, is divided uniformly among those passengers. The taxi calculates the cost (*Initial_{cost}*) for ride sharing for *P* and communicates using a *REPLY* message.

$$Initial_{cost} = \left(\frac{d1}{n1} + \frac{d2}{n2} + \cdots + \cdots \frac{d}{n}\right) * C_{km} \ldots\ldots (1)$$

In eqn. (1), *d1, d2, d3,...,d* are the distances which taxi has to travel between the pickup and drop-off location of *P* and *n1, n2, n3,...,n* is the number of passengers in a taxi who together share the distance *d1, d2, d3,...,d*, respectively. The parameter C_{km} indicates the monetary cost incurred for carrying passenger per kilometer (cost/km). When the passenger is dropped off the taxi, the cost is computed again using Eqn. (1). The re-computation is done because after communicating the initial cost to *P*, some other passengers may also have joined the ride and may have shared some distance with the existing passengers and thus they can cause the cost to be reduced further.

$$Rideshare_{cost} = \left(\frac{d1'}{n1'} + \frac{d2'}{n2'} + \cdots \ldots \frac{d'}{n'}\right) * C_{km} \ldots (2)$$

The parameter $\{d_1', d_2', \ldots, d'\}$ indicates the distance between pickup and drop-off for a single passenger and *{n1', n2',.... ,n'}* indicates the number of passengers who share this corresponding distance in the end of trip of that particular passenger. Passengers will only pay for the finally calculated cost which is possibly lower than the initially quoted price.

C. Performance Metrics

In the next section, we discuss about the performance metrics used in our system.

- ***Percentage of Requests Satisfaction (PRS):*** It is the ratio of the *ride sharing requests satisfied* by the taxi (N_s) to the *total number of ride sharing requests* (*M*) made to the taxi.

$$PRS = \left(\frac{N_s}{M}\right) * 100$$

- ***Percentage of Savings per Passenger (SPP):*** It is the ratio of *savings for a ride* (*S*) per passenger to the *initial cost for ride* ($Initial_{cost}$) per passenger. *SPP* is calculated as below.

$$S = Initial_{cost} - Rideshare_{cost}$$

$$SPP = \left(\frac{S}{Initial_{cost}}\right) * 100$$

If we calculate the savings of a passenger in terms of monetary value (S_m), then we can use: $SPP = S$.

- ***Percentage of Ride Sharing (RS):*** It is the percentage ratio of the *number of passengers whose request is satisfied* (N_r) to the *number of passengers who travelled together* in a taxi (N_p).

$$RS = (N_p/N_r) * 100$$

- ***Usability Ratio per taxi (UR):*** It is the ratio of the *time the taxi had one or more passenger* (T_{usable}) to the *total time the taxi travelled on road* (T_{total}) for a particular day.

$$UR = \frac{T_{usable}}{T_{total}}$$

- ***Relative Distance Ratio (RDR):*** The shortest-path distance between the pickup and drop-off location of a request is called *distance of a request*. RDR is the ratio of the *total distance travelled by the taxi* (D_T) to the *sum of distances of requests* (D_R) that got fulfilled with more than one passenger (on a particular day).

$$RDR = \frac{D_T}{D_R}$$

If RDR is smaller than 1, then it indicates that ride sharing actually reduces total distance travelled.

- ***Waiting Time per passenger (WT):*** It is the mean time elapsed between a time a ride sharing reqare u uest is submitted by a passenger (*t*) and the time the passenger is picked up by a taxi ($time_p$).

$$WT = time_p - t$$

D. Experimental Setup

The simulation parameters used for implementation is shown in Table IV.

TABLE IV. SIMULATION PARAMETERS

Parameters	Value
Wireless Transmission Range (T_x)	200 m
Number of Shifts for Taxis	3 (0-7, 8-15 and 16-23)
Number of Taxis in each Shift	4,000
Capacity of each Taxi (N)	4
Number of Passengers/Requests in a day	44,241
MAX_ WAIT	10, 15 (min.)
Δ (slack time)	5, 10 (min.)

We have performed our experiments using the data set of Shanghai with 44,241 passenger's requests. The requests of passengers are submitted in real time from the data set. Java is used to implement the framework. We have used Intel(R) Core (TM) (2.80GHz) processor for the experiment. Multi-threaded environment is used to simulate the taxis, i.e., each taxi represents one thread. Google Map API is used to calculate distance and time between two locations.

E. Results of Our Proposed TRS Algorithm

The number of requests fulfilled in each hour of a day is shown in Fig. 6. The total number of requests submitted on 20-02-2007 was 44,241 and out of these, 15,382 requests were fulfilled (R_f), i.e., ~ 34% of the total requests got fulfilled.

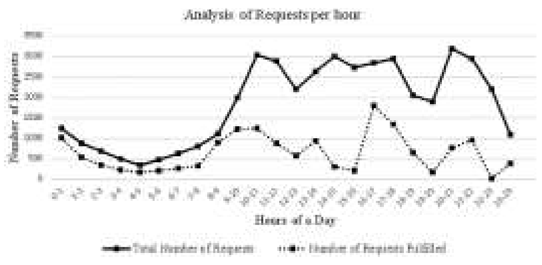


Fig. 6. Number of requests fulfilled in each hour of a day

Variation of RS with each hour of a day is shown in Fig. 7. It can be observed that, RS is highest during 7-8 am. There were 15,382 requests which got fulfilled (R_f) and out of these, 531 passengers (~3.5%) journeyed in ride sharing and remaining passengers journeyed individually.

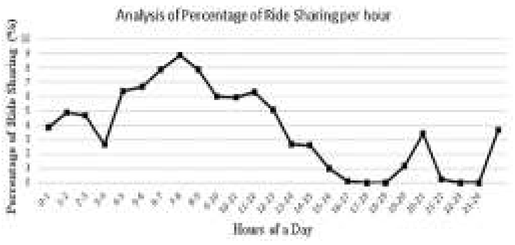


Fig. 7. Percentage of Ride Sharing per hour

Because of ride sharing, passengers have to pay less than what they have to pay without ride sharing. The savings for passengers is shown in Fig. 8.

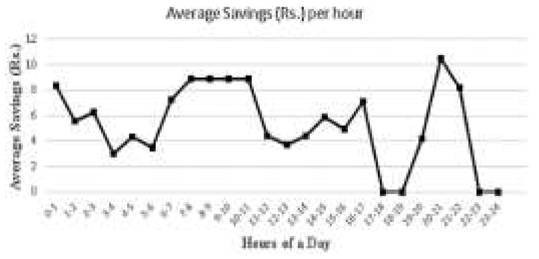
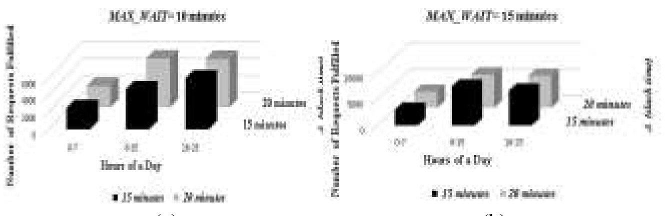


Fig. 8. Average Savings (Rs.) per hour

The average UR for a taxi is approx. 87.99% during 0-7 hour, 86.47 % during 8-15 hour and 88.89 % during 16-23 hour. So, the taxis are being efficiently utilized.

F. *Results by varying MAX_WAIT and slack time (Δ)*

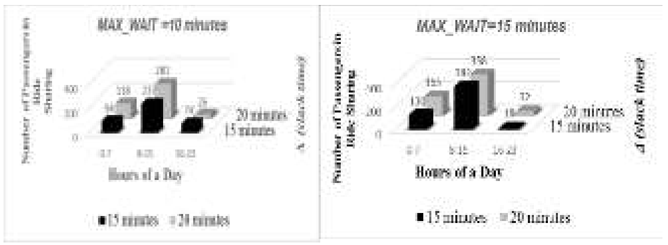
We have calculated the results using different values for MAX_WAIT and Δ. Variation of R_f with varying MAX_WAIT values are shown in Fig. 9(a) and (b), respectively. It can be observed that with increase in Δ, R_f also increases.



(a) (b)

Fig. 9. Number of Requests Fulfilled (R_f) w.r.t. MAX_WAIT and Δ

We also calculated the number of passengers who journeyed via ride sharing by changing the value of MAX_WAIT and Δ, and plotted the results in Fig. 10(a) and (b) with varying MAX_WAIT values. We get an insight that number of passengers who journeyed via ride sharing increases as the constraints are relaxed.



(a) (b)

Fig. 10. No. of Passengers in Ride Sharing w.r.t. MAX_WAIT and Δ

Fig. 11 shows that total savings per passenger (S_m) increases with increase in Δ. This happens because as Δ increases, there are more chances that a passenger event will be inserted in between the journey of other passengers, because the chances that the constraints for other existing passengers will get violated, is less.

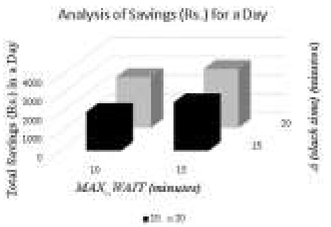
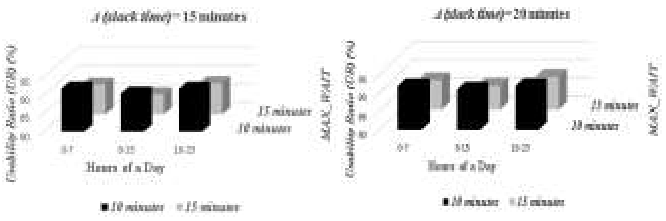


Fig. 11. Total Savings per Passenger w.r.t. MAX_WAIT and Δ

Fig. 12(a) and (b) show the variation of UR with varying Δ. We can observe that as the value of MAX_WAIT increases, UR decreases. As shown in Fig. 9, R_f increases with increase in MAX_WAIT. Thus, there are more chances that even though passenger is inserted at the end of the schedule, still passenger will accept the response from the taxi. Therefore, the total time up to which taxi travelled on road without passenger increases, which in turn, decreases the value of *UR*.



(a) (b)

Fig. 12. Usability Ratio (*UR*) w.r.t. MAX_WAIT and Δ

The variation of *RDR* w.r.to different parameters is shown in Table V. The value of *RDR* is always less than one in all the cases. Thus, the value of D_T is reduced if there is ride sharing.

TABLE V. VARIATION OF RDR W.R.TO. MAX_WAIT AND Δ

Shift of Taxi (in Hours)	MAX_WAIT=10 min		MAX_WAIT=20 min	
	Δ =15 min	Δ =20 min	Δ =15 min	Δ =20 min
0-7	0.98	0.97	0.98	0.97
8-15	0.97	0.96	0.95	0.75
16-23	0.99	0.99	0.99	0.99

G. *Comparisons of Results with Another Algorithm*

We have implemented the Taxi Distance Minimization Algorithm (henceforth called 'TDM') described in [25] for comparison purposes. The main objective of TDM is to minimize the distance travelled by taxis. For each possible pickup event insertion in taxi schedule, every possible interval for inserting drop-off event, is checked. For each of the possible schedule created after inserting pickup and drop-off events, the distance that taxi has to travel is calculated. The schedule in which distance travelled by taxi is minimum is selected as a feasible schedule and the response is sent to passenger. Let *n* (and *m*) be the number of intervals where pickup (and drop-off) event can be inserted, respectively. So, the complexity of TDM is O(n*m).

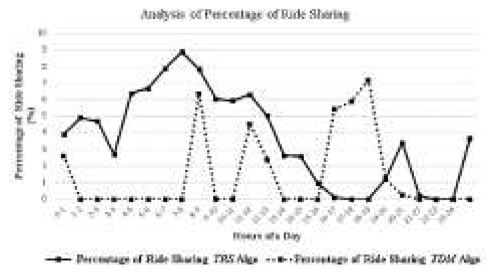


Fig. 13. Comparison based on Percentage of Ride Sharing

The variation of RS with each hour of a day using TRS and TDM are shown in Fig. 13. RS is higher for TRS except during hours 16-17, 17-18 and 18-19. In order to minimize D_T, TDM considers inserting pickup and drop-off events at such positions in a taxi schedule which may not allow ride sharing. However, TRS aims to increase RS without considering D_T. As observed in Fig. 13, RS is high during morning hour 7-8 using TRS, whereas using TDM, RS is maximum during hour 18-19.

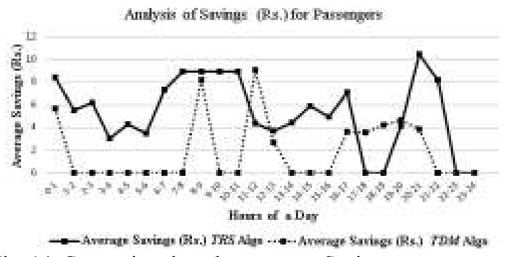


Fig. 14. Comparison based on average Savings per passenger

Fig. 14 shows the variation of S_m in each hour of a day. It can be observed that using TRS, S_m is higher during most of the hours while compared to TDM. This happens because TRS aims to increase ride sharing unlike TDM, and higher sharing (also shown in Fig. 13.) increases the S_m (savings). Similarly, variation of SPP is shown in Fig. 15 and similar trends for TRS and TDM are also observed for this case.

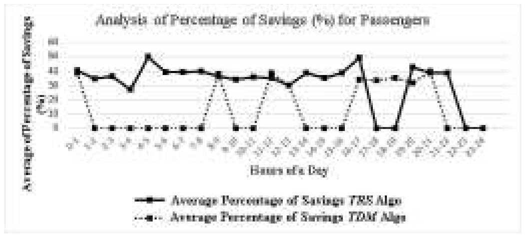


Fig. 15. Comparison based on Percentage of Savings

Though ride sharing is economical for passengers it might increase passenger waiting time. As shown in Fig. 16, TRS incurs higher WT than TDM except during hours 16-17, 17-18 and 18-19 (times during which TDM sports higher RS, as per Fig. 13).

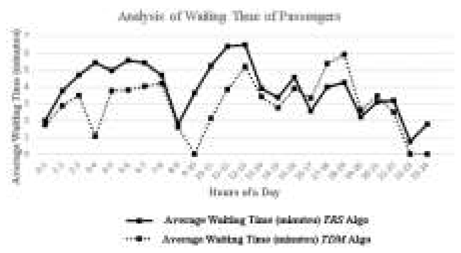


Fig. 16. Comparison based on average waiting time

We have assumed that there are three shifts for the taxis. Shift 1 is for 0-7 hours, shift 2 is for 8-15 hours and Shift 3 is for 16-23 hours. The distance travelled by taxis with passengers in each shift is plotted in Fig. 17 and we can see that taxis travelled more distances with passengers in Shifts 2 and 3 using TRS.

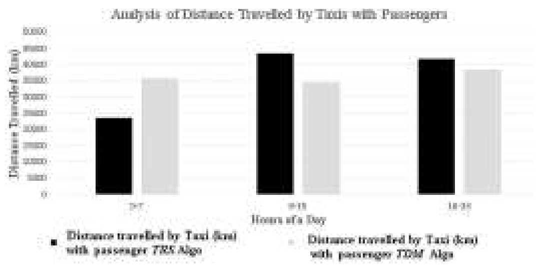


Fig. 17. Distance Travelled by Taxis with Passenger

As described in Table VI, RDR is less than 1 in each shift of the taxi using TDS. This indicates that using TDS, taxis are able to provide ride sharing and reduces D_T. However, using TDM, RDR is greater than 1 for shift 0-7 which shows that taxis using TDM are not saving on D_T by ride-sharing.

TABLE VI. COMPARISON OF RDR USING TRS AND TDM ALGORITHM

Taxi Shifts	TRS Algorithm			TDM Algorithm		
	D_T (km)	D_R in km	RDR	D_T in km	D_R in km	RDR
0-7	23173.38	23691.54	0.97	35820.64	25468.936	**1.41**
8-15	43689.12	58185.5	0.75	34519.95	40295.06	0.85
16-23	41901.1	42031.84	0.99	38514.13	39435.72	0.97

VI. CONCLUSION AND FUTURE WORKS

In this paper, we proposed a linear time distributed taxi ride sharing (*TRS*) algorithm, which aims to increase taxi occupancy and profit for drivers and savings for passengers. Our algorithm also reduces the total distance travelled by taxis as passengers share ride for the common path. Empirical results using a large scale taxi GPS traces from Shanghai, PRC shows that TRS algorithm can greatly outperform a Taxi Distance Minimization (TDM) algorithm [25]. TRS algorithm accommodates 33% higher ride share among passengers while dealing with 44,241 requests handled by 4,000 taxis on a single day in Shanghai. In future, we plan to extend our algorithm to incorporate rider matching based on gender, friendship or interest factors.

VII. ACKNOWLEDGMENT

Research work reported in this paper is partially funded by the Alexander von Humboldt Foundation through the post-doctoral Fellow Dr. Vaskar Raychoudhury. Also, we acknowledge the critical comments and advices provided by Dr. Rajdeep Niyogi, Associate Professor, Department of CSE, IIT Roorkee, India, during the initial stages of this work. We also thank Ms Shrawani Silwal from Miami Univ., Ohio, USA for her help to improve the paper.

REFERENCES

[1] J.-F. Cordeau and G. Laporte, "The dial-a-ride problem: models and algorithms," Annals of Operations Research, vol. 153, no. 1, pp. 29-46, 2007.

[2] J.-F. Cordeau, "A branch-and-cut algorithm for the dial-a-ride problem," Operations Research, vol. 54, pp. 573-586, 2006.

[3] L.M. Hvattum, et al., "A branch-and-regret heuristic for stochastic and dynamic vehicle routing problems," Networks, vol. 49, pp. 330-340, 2007.

[4] J-F. Cordeau and G. Laporte, "A tabu search heuristic for the static multivehicle dial-a-ride problem," Transportation Research Part B: Methodological, vol. 37, pp. 579-594, 2003.

[5] Z. Xiang, C. Chu, and H. Chen, "A fast heuristic for solving a largescale static dial-a-ride problem under complex constraints," European Journal of Operational Research, vol. 174, pp. 1117–1139, 2006.

[6] 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, pp. 377-387, 2004.

[7] R. Baldacci, V. Maniezzo, and A. Mingozzi,, "An exact method for the car pooling problem based on lagrangean column generation," Operations Research, vol. 52, pp. 422-439, 2004.

[8] R. W. Calvo, et al., "A distributed geographic information system for the daily car pooling problem," Computers & Operations Research, vol. 31, pp. 2263-78, 2004.

[9] "Slugging," 2017. [Online]. Available: https://en.wikipedia.org/wiki/Slugging. [Accessed 24 April 2017].

[10] N. Agatz, et al., "Optimization for dynamic ride-sharing: A review," European Journal of Operational Research, vol. 223, pp. 295-303, 2012.

[11] M. Rigby, et al., "An opportunistic client user interface to support centralized ride share planning," in Proc. of the 21st ACM SIGSPATIAL Int'l Conf. on advances in geographic information systems, 2013.

[12] S. Ma, Y. Zheng, and O. Wolfson, "Real-Time City-Scale Taxi Ridesharing," IEEE Transactions on Knowledge And Data Engineering, July 2015.

[13] B. Shen, Y. Huang, and Y. Zhao, "Dynamic Ridesharing," SIGSPATIAL Special, vol. 7, no. 3, pp. 3-10, 2016.

[14] C.-C. Tao, "Dynamic Taxi-sharing Service Using Intelligent Transportation System Technologies," in International Conference on Wireless Communications, Networking and Mobile Computing, 2007.

[15] S. Ma, Y. Zheng, and O. Wolfson, "T-Share: A Large-Scale Dynamic Taxi Ridesharing Service," in Data Engineering (ICDE), 2013.

[16] Y. Huang, F. Bastani, R. Jin, and X. S. Wang,"Large Scale Real-time Ridesharing with Service Guarantee on Road Network," Proceedings of the VLDB Endowment, vol. 7, no. 14, pp. 2017-2028, 2014.

[17] C. Tian, Y. Huang, Z. Liu, F. Bastani, and R. Jin, "Noah: A Dynamic Ridesharing System," in International Conference on Management of Data, 2013.

[18] P.-Y. Chen, J.-W. Liu, and W.-T. Chen, "A Fuel-Saving and Pollution-Reducing Dynamic Taxi-Sharing Protocol in VANETs," in Vehicular Technology Conference Fall (VTC 2010-Fall), 2010.

[19] M. Ota, H. Vo, C. Silva, and J. Freire, "A scalable approach for data-driven taxi ride-sharing simulation," in IEEE International Conference on Big Data (Big Data), 2015.

[20] D. Dimitrijevi, N. Nedi, and V. Dimitrieski, "Real-time carpooling and ride-sharing: Position paper on design concepts, distribution and cloud computing strategies," in 2013 Federated Conference on Computer Science and Information Systems, 2013.

[21] Y. Lin and H. Shen, "Vshare: A wireless social network aided vehicle sharing system using hierarchical cloud architecture," in Internet-of-Things Design and Implementation (IoTDI), 2016 IEEE First International Conference on. IEEE, 2016, pp. 37-48.

[22] W. M. Herbawi and M. Weber, "A Genetic and Insertion Heuristic Algorithm for Solving the Dynamic Ridematching Problem with Time Windows," in Genetic and Evolutionary Computation. ACM, 2012, pp. 385-392.

[23] S. Varone and V. Janilionis, "Insertion Heuristic for a dynamic Dial-A-Ride Problem Using Geographical Maps," in MOSIM 2014, 10ème Conférence Francophone de Modélisation, Optimization et Simulation, Nov 2014, Nancy, France, 2014.

[24] D. O. Santos and E. C. Xavier, Dynamic Taxi and Ridesharing: A Framework and Heuristics for the Optimization Problem., in IJCAI, vol. 13, 2013, pp. 2885-2891.

[25] B. Coltin and M. Veloso, "Ridesharing with passenger transfers," in Intelligent Robots and Systems (IROS 2014), 2014 IEEE/RSJ International Conference on. IEEE, 2014, pp. 3278-3283.

[26] F. Bistaffa, A. Farinelli, and S. D. Ramchurn, "Sharing Rides with Friends: A Coalition Formation Algorithm for Ridesharing," in Twenty-Ninth AAAI Conference on Artificial Intelligence, 2015.

[27] B. Cici, et al., "Assessing thePotential of Ride-sharing Using Mobile and Social Data: A Tale of Four Cities,"in Proceedings of the 2014 ACM International Joint Conference on Pervasiveand Ubiquitous Computing. ACM, 2014, pp. 201-211.

[28] G. Gidofalvi, el al., "Highly scalable trip grouping for large-scale collective transportation systems," in Proceedings of the 11th international conference on Extending database technology: Advances in database technology. ACM, 2008, pp. 678-689.

[29] P. M. d'Orey, et al., "Empirical evaluation of a dynamic and distributed taxi-sharing system," in 15th International IEEE Conference on Intelligent Transportation Systems, 2012.

[30] S. Winter, "Intelligent Self-Organizing Transport," KI, vol. 22, pp. 25-28, 2008.

[31] W. Zhao, et al., "Social group architecture based distributed ride-sharing service in VANET," International Journal of Distributed Sensor Networks, vol. 10, 2014.

[32] A. Arora, et al., "Automated ride share selection using vehicular area networks," in IEEE ICC Communications Workshops, 2009, pp. 1-6.

[33] S. Winter and S. Nittel, "Ad hoc shared-ride trip planning by mobile geosensor networks," International Journal of Geographical Information Science, vol. 20, no. 8, pp. 899-916, 2006.

[34] M. Sghaier, et al., "A distributed dijkstra's algorithm for the implementation of a Real Time Carpooling Service with an optimized aspect on siblings," in 13th International IEEE Conference on Intelligent Transportation Systems, 2010, pp. 795-800.

[35] R. Geisberger, et al., "Fast detour computation for ride sharing," arXiv preprint arXiv:0907.5269, 2009.

[36] Cao, B., Alarabi, L., F. Mokbel, M., Basalamah, "SHAREK: A Scalable Dynamic Ride Sharing System," in 16th IEEE International Conference on Mobile Data Management, 2015.

[37] C.-C. Tao, "Dynamic taxi-sharing service using intelligent transportation system technologies," in IEEE International Conference on Wireless Communications, Networking and Mobile Computing, 2007, pp. 3209-3212.

[38] SUVnet-Trace data (http://wirelesslab.sjtu.edu.cn/taxi_trace_data.html)