A Dynamic Taxi Ride Sharing System Using Particle Swarm Optimization

Shrawani Silwal
Dept. of CSE
Miami University
Oxford, OH, USA
silwals@miamioh.edu

Vaskar Raychoudhury
Dept. of CSE
Miami University
Oxford, OH, USA
raychov@miamioh.edu

Snehanshu Saha
Dept. of CS&IS & APPCAIR
BITS Pilani Goa Campus
Goa, India
scibase.snehanshu@gmail.com

Md Osman Gani
Information Systems
University of Maryland,
Baltimore County, MD, USA
mogani@umbc.com

Abstract—With the rapid growth of on-demand taxi services, like Uber, Lyft, etc., urban public transportation scenario is shifting towards a personalized transportation choice for most commuters. While taxi rides are comfortable and time efficient, they often lead to higher cost and road congestion due to lower overall occupancy than bigger vehicles. One efficient way to improve taxi occupancy is to adopt ride sharing. Existing ride sharing solutions are mostly centralized and proprietary. Moreover, given the wide spatio-temporal variation of incoming ride requests designing a dynamic and distributed shared-ride scheduling system is NP-hard. In this paper, we have proposed a publisher (passengers) and subscriber (taxis) based ride sharing system that provides effective real-time ride scheduling for multiple passengers. A particle swarm based route optimization strategy has been applied to determine the most preferable route for passengers. Empirical analysis using large scale single-user taxi ride records from Chicago Transit Authority, show that, our proposed system, ensures a maximum of 91.74% and 63.29% overall success rates during non-peak and peak hours, respectively.

Index Terms—Dynamic Ride Sharing, Shared Transportation, Optimization, Evolutionary Algorithms

I. INTRODUCTION

Emerging technologies like on-demand taxi services and autonomous vehicles are soon to change the very concept of car ownership and result in a more efficient public transportation system. As per the National Household Travel Survey [1] in the USA, the average vehicle occupancy (as of 2017) is approximately 1.6 and has not improved since 2009. Similar trend is observed in multiple major cities around the world such as Chicago [2], New York [3], San Francisco [4], Shanghai [5], etc. As reported in the Chicago Data portal [2] the traffic congestion in Chicago has increased the average travel time by 26%, amounting to an average 29 minutes of extra commuting time per day and can reach up to a maximum of 90 minutes. Surely, the increase is attributed to the increasing number of personal vehicles. So, one way to reduce the road traffic is to improve vehicle occupancy and in this paper we look into the ride-sharing problem of taxicabs. Ride-sharing aims to bring together multiple riders sharing a taxi ride in full or in part with certain amounts of detour to pick or drop passengers [6]. Besides reducing the road traffic, it can reduce the overall carbon footprint and optimize the revenue of taxi drivers [7].

Ride-sharing has grown into a widely researched topic in recent years. Multiple applications like UberPool1, Lyft2 and DiDi3 offer ride-sharing options to users. However, most of them are proprietary systems and operates in a centralized manner. They hardly consider adapting taxi routes in real-time to accommodate the incoming ride requests arriving in an online manner.

The ride-sharing problem originated from the classic Dial-a-Ride Problem (DARP) [8] which is \( NP \)-hard and can have multi-objective extensions. In DARP, the passengers request rides to travel between specific origins and destinations, which are fulfilled by either a single vehicle or multiple vehicles. The combined goal is to find a minimum cost route for the vehicles to transport all the passengers. Owing to the non-deterministic nature of DARP, many heuristic and meta-heuristic algorithms are developed to solve it. However, the spatio-temporal nature of modern on-demand ride-sharing systems has been grossly overlooked by the DARP problem. Also, many existing systems use simulated data or bare minimal real data to test the performance which may result in false positives.

In this paper, a dynamic taxi ride-sharing system is proposed to handle the incoming ride requests in

1Uber: https://www.uber.com/
2Lyft: https://www.lyft.com/
3DiDi: https://www.didiglobal.com/
a real-time manner. We used a Particle Swarm based meta-heuristic optimization algorithm (PSO) for ride matching. The contributions are summarized as follows:

- We developed the first-ever PSO-based optimization framework for solving the dynamic taxi ride sharing problem. The problem is formulated as a constrained single objective optimization problem.
- We analyzed large-scale single ride taxi trip data from Chicago to identify the spatio-temporal traffic patterns and trends. We then decided to use the Chicago taxi trip dataset for the evaluation of our proposed algorithm.
- We defined benchmark performance metrics and our experiment results show that our proposed algorithm can achieve a ride sharing success rate up to 91.74% even in the presence of spatio-temporal coordination challenges.

II. RELATED WORK

In our previous work we have shown a detailed classification of Ride Sharing Systems (RSS) [9] depending on the nature of the problem (static vs dynamic) and the system architecture adopted (centralized, distributed and hybrid). In this Section, we mainly discuss the various distributed and dynamic ridesharing systems and the different optimization models adopted by them.

The features, constraints, and objectives of a static and dynamic ride-sharing problem were highlighted in details by Agatz et al. [10]. They briefly highlighted the dynamic addition of drivers and riders in the system. The most prevalent type of dynamic ride-sharing based on smart devices involve multiple drivers and riders. In such systems, making suitable matches as well as deciding on optimal routes is complex. This paper explores the challenges for developing a dynamic system in depth. However, their study does not include an implementation of the dynamic system.

D’Orey et al. [11], proposed a novel dynamic and distributed taxi-sharing system. Their goal was to design a flexible system for both passengers and taxis which was cost effective. They assert that by making the system distributed, i.e., performing the complicated computations in each vehicle level rather than at a central location, a smaller processing time is possible to achieve. The taxi and passengers make use of wireless technology to send and receive requests and responses. On receiving a new request, a permutation of all the routes are computed and filtered based on the constraints. The passengers receive the cost estimation when all the constraints are met. The algorithm at the passenger side can select the response having minimum cost. The system was tested with a small set of simulated data but no real dataset was used.

Bathla et al. [5] proposed a Taxi Ride Sharing (TRS) algorithm similar to d’Orey et al. [11] and it is a distributed algorithm that used localized communication between passengers and nearby taxi drivers. It can handle real-time ride requests without disturbing the confirmed passenger trip schedules. They have tested the algorithm with a large scale GPS traces of 4,000 taxis in Shanghai, China. They identified the spatio-temporal challenges of a dynamic ride sharing system. Their evaluation of the system is based on improvement in driver’s profit and passenger’s cost savings while reducing the total distance travele (while considered without ride-sharing). This paper has similar goals to the current study such as an increase in shared rides for the taxis. However, they did not model their problem using optimization approach.

Lokhandwala and Cai [12] proposed an agent-based model having a matching algorithm between taxi and passengers. They modeled using traditional taxis as well as autonomous vehicles including the complexities of real-world scenarios and road infrastructures. They included a number of added preferences such as maximum deviation allowed, sharing participation level and also modeled the changing shifts in traditional taxis. This type of taxi shift change is applicable for traditional taxis and does not necessarily apply to the modern app-based on-demand ride-sharing systems. Their RSS is evaluated based on fleet reduction, which is reducing the total number of the taxi, by increasing the resource (the taxi space) utilization. Improvement is noticed which adds to the environmental benefits as well. They claim that the system was tested up to 8000 fleets (taxi) using New York taxi data set.

Hsieh et al. [13] modeled the carpooling problem as an integer programming problem. Their approach is very close to the proposed work, as the problem is framed in a similar way using evolutionary (meta-heuristic) technique. They developed a discrete co-evolving particle swarm optimization (DCCPSO) algorithm, a meta-heuristic approach to solve the single objective function. However, they test the system using small numbers of taxis and passengers.

III. DATA SET ANALYSIS

We have used the taxi trips data set from the Chicago Data Portal for our experiments.

Chicago Data Portal: https://data.cityofchicago.org/Transportation/Taxi-Trips/wrvz-psew
A. Temporal Analysis of Chicago Taxi Data

We performed a temporal analysis of the passenger requests from the Chicago taxi dataset (Figure 1(a)) and found distinctive peak and non-peak hour patterns. We have defined non-peak hours as the hour during which the number of ride requests are less than 1000 or the lowest within the 24-hour-period. From the figures, we can identify non-peak hours between 2-7 AM consistently on weekdays. However the patterns change over the weekends, and we notice non-peak hours between 6-8 AM. The difference in hours suggests that on weekdays, people prefer to rest over the night and work during the day. While during weekdays people most likely start their day late and prefer staying in during the morning hours. Similarly, we identified the peak hours during the weekdays typically between 8-9 am, 12-2 pm and 4-7 pm. The peak hours are the higher numbers of requests observed during the 24 hour of the day. On Saturdays nights the peak hours were seen between 12-2 am and 10-12 pm. While on Sundays, we noticed peak hours between 12-2 am. These trends point to the fact that people are likely to stay out till late at night on weekends compared to weekdays.

B. Spatial Analysis of Chicago Taxi Data

We performed a spatial analysis of the Chicago dataset (Figure 1(b)) to understand the distribution of requests throughout the 77 Community Areas of the Chicago metropolitan area. On close examination, we found the ride request distribution is highly uneven. A high concentration of ride is originating from and terminating into only 3-6 community areas. A very high number of rides are found in the Chicago downtown area, while the distribution is sparse on the other areas of the city. An outlier can be observed at Chicago O’Hare airport which also seems to experience a large number of requests.

IV. RIDE SHARING PROBLEM AND PROPOSED ALGORITHMS

Given a set of passengers $P = \{p_1, p_2, ..., p_n\}$ and a set of taxis $T = \{t_1, t_2, ..., t_m\}$ each having a capacity of 4, the taxi ride sharing problem aims to find a set of passengers who can share the taxi ride for an entire journey or the part thereof. Since the ride requests keep arriving in real-time, it makes the decision making at the taxi-end non-trivial. A hybrid publish/subscribe based communication model was chosen for the system design. The ride requests are channeled into a queue, from which they are distributed to the taxis in the vicinity (within a radius of 2 KMs/3 KMs). The variables and messages used in our algorithm are discussed in the Table I and Table II. We have made the following assumptions:

1) The execution of algorithms at taxi or passenger-end are faster than the average taxi speed.
2) Every incoming REQUEST for ride sharing is considered independent of each other.
3) Each taxi can participate in a single passenger REQUEST computation at a time.
4) The pickup event timestamp of a passenger should always precede the drop-off event timestamp for the same REQUEST.

| TABLE I: Variable Definitions |
|--------------------------------|
| Variables | Significance |
| $P$ | Set of Passengers , passenger $p \in \{1, 2, 3, ..., P\}$ |
| $T$ | Set of Taxi, driver $t \in \{1, 2, 3, ..., T\}$ |
| $S_p$ | Origin(source) location for $p$ |
| $D_p$ | Destination for $p$ |
| $t_{w_p}$ | Time at which $p$ made a ride-sharing request |
| $t_{rej_p}$ | Time at which $p$ received a response |
| $t_{pick_p}$ | Time at which passenger $p$ is picked up by taxi |
| $t_{cap}$ | Capacity of taxi $t$ |
| $w_{t_p}$ | Waiting Time after sending request |
| $Max wait$ | Maximum Wait Time |
| $\Delta$ | Detour tolerance |
| $d_{src}$ | Time to Travel from source to destination |
| $x_p$ | \{ 1. if passenger $p$ is picked up \\
| | 0. otherwise \} |
| $R_p$ | Request from a passenger $p$ |
| $MR_p$ | Request matched to a Taxi $t$ |
| $CR_p$ | Confirmation of Request($R_p$) from passenger $p$ |
| $d_{det}$ | Detour distance for $t$ |
| $AckStatus$ | Acknowledgement status sent by taxi to passenger |
| $C_t$ | Estimated cost of the taxi ride |
| $num_p$ | number of passengers in a taxi $t$ |
| $t_{messages}$ | total no of requests/response/confirm and ack messages |

| TABLE II: Messages Used in the Algorithm |
|-----------------------------------------|
| Messages | Significance |
| REQUEST($p_1, S_p, D_p, t_{w_p}$) | Request for ride-sharing sent by Passenger ($P$) |
| REPLY($T_{id}, t_{pick}, S_{drop}, C_p$) | Reply from $T$ to $P$ with estimated pick-up time, drop off time and cost for the ride |
| CONFIRM($T_{id}, t_{pick}$) | Acceptance/Rejection status sent by $P$ to $T$ |
| ACR($T_{id}, P_{id}, AckStatus$) | Acceptance/Rejection status sent by $P$ to $T$ |

A. Algorithm at Passenger-end

At passenger-end (Algorithm 1), the REQUEST message is published to the message queue and is routed by the request router to all the taxis in the wireless range (2 or 3 KMs). The waiting timer $w_{t_p}$ is initialized and until it expires, all the REPLY messages are collected in a queue. Then, the best response is selected based
on the minimum cost, the number of passengers already assigned to the taxi (higher number preferred), and the time and distance to reach from source to destination. After the response is selected a CONFIRMATION message is sent to the taxi. If no REPLY was received, a new REQUEST has to be published.

**Algorithm 1** Algorithm at Passenger-end

1. Publish REQUEST message
2. Start the timer $w_{R_p}$
3. When $w_{R_p} \geq Max_{wait}$
4. if REPLY queue size $> 0$ then
5. Select the best response from REPLY queue
6. Send CONFIRMATION message to the taxi
7. else
8. No taxi Available, send new REQUEST
9. end if

**Algorithm 2** Algorithm at Taxi-end

1. Taxi receives REQUEST from passenger P
2. if $t_{seats} < 4$ then
3. Call Algorithm 3 to schedule the route
4. else
5. Return response, Taxi is not available
6. end if
7. Send response REPLY to P
8. Confirmation timer $T_{confirm}$ starts
9. if $T_{confirm} \leq 90$ secs then
10. if CONFIRMATION is received from P then
11. Update Permanent Schedule and Occupancy
12. end if
13. end if

**B. Algorithm at Taxi-end**

At the taxi-end (Algorithm 2), when a REQUEST message is received, the processing begins only if the taxi’s maximum capacity is not yet reached. If the taxi can accommodate the incoming request, the feasibility of including the new request into the taxi’s schedule is done by making a call to PSO for Route Optimization (Algorithm 3). After the route is scheduled using Algorithm 3, the taxi sends a REPLY back to the passenger and starts the confirmation timer $T_{confirm}$ which expires in 90 seconds. In case a confirmation is received within time, the permanent schedule of the taxi and the taxi occupancy is updated. Once the timer expires, the taxi can process new requests.

**C. PSO for Route Optimization**

Particle Swarm Optimization (PSO) is an evolutionary strategy inspired by the behavior of flock of animals or birds. It helps in attaining the global minima of a given function. It works iteratively to converge a population of randomly initialized solutions also known as the particles, towards a globally optimal solution (global-best solution). Each particle within the population continues to track its current position and the best solution it has met, known as $p_{best}$. These particles have an associated velocity to span across the search space. The overall best solution of the swarm is called $g_{best}$, which is kept track of by the swarm itself. During each iteration, the swarm updates the velocity of the particle towards its $p_{best}$ and $g_{best}$ values.

In an effective optimization setting, we search for global optima in the search space. The fitness of the function needs to be improved in an iterative search process for single or multi-objective optimization problems. Let $f(x)$ be the function to be minimized. In our case, the fitness function is the Passenger Travel Time($tt_{sd}$). Thus, the problem is one of single objective optimization constrained by conditions such as taxi capacity, waiting time etc. Passenger Travel Time is the total time taken...
for all the passenger to reach from source to destination including the detour caused by sharing the taxi with other passengers.

$$\min t_{sd} = \sum_{p=1}^{p} x_p \text{time}(S_p, D_p)$$  \hspace{1cm} (1)$$

The objective functions are subject to all the constraints:

$$t_{cap} \leq 4, \forall t \in \{1, 2, 3, ..., T\}$$  \hspace{1cm} (2)$$

$$w_{R_p} \leq \text{Max wait}, \forall p \in \{1, 2, 3, ..., P\}$$  \hspace{1cm} (3)$$

$$tt_{sd} \leq \text{time}(S_p, D_p) + \Delta, \forall p \in \{1, 2, 3, ..., P\}$$  \hspace{1cm} (4)$$

These are three constraints (2, 3, 4), defined above for the objective functions and together form a non-convex/non-concave optimization problem. $t_{cap}$ is the condition for checking the taxi capacity, which is set to less than or equal to 4. Equation 3 defines the waiting time $w_{R_p}$, which is the waiting time after the passengers requests for a ride and receives a response. This value should be smaller than the maximum wait time ($\text{Max wait}$) which is predefined (for example 5, 10 minutes). Lastly, the time taken to travel($tt_{sd}$) from source to destination for any passenger, should not exceed the original travel time without sharing plus the detour tolerance($\Delta$).

PSO is characterized by a population of particles in space, which aim to converge to an optimal point. The movement of the particles in space is characterized by two equations, namely, velocity and position update equations, which are as follows:

$$v_{i}^{t+1} = \omega v_{i}^{t} + c_1 r_1(p_{i}^{best} - x_{i}^{t}) + c_2 r_2(g_{i}^{best} - x_{i}^{t})$$  \hspace{1cm} (5)$$

$$x_{i}^{t+1} = x_{i}^{t} + v_{i}^{t+1}$$  \hspace{1cm} (6)$$

where $\omega, c_1, c_2 \geq 0$. Here, $x_{i}^{t}$ refers to the position of particle i in the search space at time t, $v_{i}^{t}$ refers to the velocity of particle i at time t, $p_{i}^{best}$ is the personal best position of particle i, $g_{i}^{best}$ is the best position amongst all the particles of the population.

For each particle, two random numbers ($r_1$ and $r_2$) are chosen from a uniform distribution that lies between 0 and 1. Omega $\omega$ is the inertia weight and $c_1$, $c_2$ are the cognitive coefficients, which defines how much a particle can rely on its experience and the experience of its neighbors. In other words $c_1$, $c_2$ are the learning rates. In case a new position of the particle is a better fit than its $p_{best}$ value, the algorithm updates the $p_{best}$ to the new position. Once these iterations of $p_{best}$ values are completed, the algorithm converges to a overall optimal or $g_{best}$ value. The algorithm is finally terminated when the $g_{best}$ has not changed after a set of specified iterations. The parameters to tune for optimal performance are Number of particles (swarm size), $c_1$ (importance of personal best) $c_2$ (importance of neighbourhood best) and $v_{max}$ (limit on velocity). In other words, $V_{max}$ is a maximum velocity, which is user-defined value to maintain a threshold on particle’s velocities. Driven purely by empiricism, number of particles (10—50) are reported as usually sufficient. Usually $C1 + C2 = 4$, stands by the same logic. The $C1 + C2 = 4$ also satisfies stability condition of particle velocities. If $v_{max}$ is too low, the convergence is too slow and too high a value of $v_{max}$ makes the algorithm unstable. The algorithm 3 describes the process in various steps.

**Algorithm 3 PSO for Route Optimization**

1: Current schedule of the Taxi $Q_{temp}$ ← Events from Request
2: Initialize PSO parameters and particles using $Q_{temp}$
3: Set random position and velocity for every particle
4: do
5: Find every particle’s fitness value ($Fit_{xr}$) using the function (Eq 1)
6: foreach particle p do
7: if $Fit_{xr} < p_{best}$ then
8: $p_{best}$ ← $Fit_{xr}$ & save vehicle schedule
9: end if
10: if $Fit_{xr} < g_{best}$ then
11: end if
12: Calculate particle velocity (Eq. 5)
13: Update particle position (Eq. 6)
14: Generate new solution for particle
15: end foreach
16: while the maximum number of iterations are not reached

V. PERFORMANCE MEASUREMENTS

A. Experimental Setup

The simulation parameters chosen for the experiments are listed in Table III.

| Parameters               | Values  |
|--------------------------|---------|
| Capacity of each Taxi    | 4       |
| Taxi search radius (passenger-centric) | 2 km, 3 km |
| $\Delta$ (Detour Tolerance) | 10 minutes |
| Particle Count           | 100     |
| Maximum Number of Epochs | 100     |

We simulated our algorithm using Java and carried out the data analysis using Python. We used two machines
for the simulation. One of the machine is an Intel(R) Core(TM) i7-3770 CPU (3.40GHz) Linux machine (version 4.15.0-52-generic) with 12 GB RAM. The other machine is an Intel(R) Xeon(R) W-2133 CPU (3.60GHz) Windows machine with 128 GB RAM. Apache Kafka is used for message passing and Redis6 is used for in-memory data storage. Routing APIs [14] [15] were used to calculate the distance and time between the source and destinations. POSTMAN [16] is used for posting ride requests and registering taxis. The taxi clients, passenger clients, and router request applications were simulated as multiple processes running on different ports in these two machines.

We performed a simulation using real-world data from the Chicago Taxi dataset [2] as introduced in Section III. For each day of the week, peak (6–7 PM) and non-peak (4–5 AM) hour data were extracted. We observed different peak and non-peak hours during the weekends. During the non-peak hour merely 100–250 ride requests were submitted, while, during peak hour, 1900–4200 ride requests were submitted.

For our experiments, we used the original set of taxis that served those requests as well as injected additional taxis in various spots defined as taxi sets 1 through 4. The additional taxis were determined by spatial analysis shown in Section III. Any spot with more than five requests originating at any point in time was marked as spots with higher demand. Whereas spots with less than five requests at any time were marked as spots of lower demand. Based on this principle, we prepared four sets of experimental taxi data with varying numbers:

- **Taxi Set 1:** Considers only the taxis ($T_{oq}$) that originally served the requests in the dataset.
- **Taxi Set 2:** Consists taxis in Set 1 and additional ones injected into the spots of low demands. For every such spot, one taxi was added.
- **Taxi Set 3:** Considers taxis in Set 1 and additional ones injected into the spots of high demands. For every such spot, one taxi was added. This set has more taxis compared to Set 1 and 2.
- **Taxi Set 4:** Considers twice the taxis ($T_{og}$) in Set 1. This was the largest of four sets.

The additional taxis can either help to improve the performance of the system and reduce it further. By carrying out experiments with these separate sets, we would be able to draw more definite results on our RSS’s behavior. The experiments were performed using the four sets of taxis described above with a variation in their search radius of 2 km and 3 km. We performed experiments for every day of a week and closely monitored the behavior of the ride-sharing system. To select appropriate values for particle count and the maximum number of epochs for the PSO algorithm, we experimented with different values before choosing the ones in the Table III.

**B. Performance Metrics**

The results were calculated based on the metrics designed for the proposed system for each run (non-peak hour and peak hour).

1) **Overall Success Rate of Ride Matching (OSR)**

The ratio of the total number of rides matched to the taxis to the total number of ride requests made by the passengers. OSR also includes the taxis that are matched to pick only a single passenger.

2) **Success Rate of $N^{th}$ Passenger (SRNP)**

Defined as the success rate of matching first, second, third and fourth ($N = \{1, 2, 3, 4\}$) passengers to the taxis. This helps to visualize the distribution of requests among available taxis in the ride-sharing system.

3) **Distance Ratio (DR)**

The ratio of the sum of individual trip distances between pick-up and drop-off locations for all passengers to the total distance traveled with more than one passenger.

4) **Average Passenger Waiting Time (AWT)**

The average difference in time between the instant a passenger request is submitted and the instant a taxi picks up the passenger.

5) **Average number of messages (ANM)**

It is defined as the ratio of the total number of messages that are sent and received in the ride-sharing system to the total number of confirmed (successful) ride requests for an hour.

6) **Average Request Processing Time (ARPT)**

The average difference in time between the instant a passenger ride request is submitted and the instant a response from the ride-sharing system is received. This wait time is less than the maximum wait ($\text{Max}_{\text{wait}}$) time.

**C. Results and Analysis**

Experiments were conducted separately for the non-peak hours and peak hours for every day of a week in summer (July 10th - July 16th 2017) and a week in winter (Dec 12th - Dec 18th) as shown in the following sections. Due to space constrains, we only discuss the results for the summer week here and identical patterns for the winter week can be found in [17].

---

5 Apache Kafka: https://kafka.apache.org/

6 Redis: https://redis.io/
We recorded the results for both peak and non-peak hours in three groups, i.e., weekdays (Monday - Thursday), weekends (Friday and Saturday) and Sunday.

The overall success rate (OSR) for the non-peak hour is shown in Fig 2.I(a)-(c). Fig 2.I(a) is the success rate for weekdays, the maximum value is 97.05\%, which is obtained for taxi set 2 for a taxi radius of 3 km. Taxi set 1 also performs efficiently with 92.53\%. As the number of taxis increased gradually from taxi set 1 to 4, we observe that the OSR increases. The OSR also increases with a wider taxi search range of 3 km, as the ride request can reach a larger number of taxis. Similarly, the ride-sharing pattern for the weekend is captured in Fig 2.I(b). We can see that the OSR is significantly improved for taxi set 3 which is observed at 99.45\%. The results are improved by injecting additional taxis in the spots of more demands. The OSR during Sunday differs from the weekends’ (Fri-Sat) patterns, as seen in Fig 2.II(c). The injection of additional taxis in taxi sets 2, 3 and 4 does not improve the OSR.

The results for the peak hour as shown in Fig 2(d)-(f). It can be observed that the performance of the RSS during peak hours is not as good as that during non-peak hours. This is due to the an extremely large number of incoming requests. From Fig 2.I(d) we observe the OSR to be its maximum with 65.05\% for taxi set 3. Similarly, for weekends (2.II(e)) and Sunday (2.II(f)), the overall success rate during peak hour improves on injecting more taxis in the spots of more demands. However, the OSR suffers by increasing the taxi search radius from 2 km to 3 km, as observed in the figure throughout. A larger taxi radius sends the ride-sharing request to more number of taxis requiring additional computations for ride matching, thus deteriorating the performance.

Fig 2.VI and 2.VII show the success rate of N\textsuperscript{th} passenger (SRNP). Through these figures we can understand how the confirmed rides are allotted to the taxis. The figures 2.VI(a)-(c) and 2.VII(a)-(c) are for non-peak hours and we can see that the highest percentage is with 1 passenger. This pattern is similar for both 2 km and 3 km radius. This arises due to sporadic requests during non-peak hours, the RSS tries to match at least one passenger. We see that there is not a very large gap, the remaining rides are almost equally (above 20\%) shared by taxis having 2, 3 and 4 passengers as well. In contrast the behaviour of our RSS varies for the peak hours (figures 2.VI(d)-(f) and 2.VII(d)-(f)). We can see that, less rides are allotted to taxis with only one passenger. The ride match for 1 passenger only in a taxi is as low as 11.99\%. We can also see for the other case i.e., with 2, 3 and 4 passengers, the OSR is almost similar. This shows that during the peak hours the taxis already occupied by passengers are likely to accept incoming requests. This is because of the fact that, during peak hours more taxis as well are passengers are likely to be present in common locations and are easily matched.

The results for distance Ratio (DR) is captured in Figure 2.II(a)-(c) for the non-peak hours. We notice a consistent behaviour for all the three categories as well as the different combination of taxi sets with 2 km and 3 km radius. Higher DR value signifies saving in total distance travelled using ride sharing compared to the total distance travelled for all individual trips. For both weekdays 2.II(a) and weekends 2.II(b), the DR value is consistently better for taxi search radius within 2 km. We can infer that if the taxis are nearer to the ride requests, there will be smaller deviation from their original path. Thus we obtain a better result for 2 km taxi search radius. For Sundays, as seen in 2.II(c), the behaviour is consistent for taxi sets 1, 2 and 3, however for taxi set 4, the DR for 3 km performs better. The reason can be due to better positioning of taxis and similar routes for the data set on the particular date. The results for the peak hours (Figure 2.II(d)-(f)) are consistent with our observations for non-peak hours. However, the pattern for Sunday is not consistent with our previous inferences as seen in Figure 2.II(c). This is inferred to be caused by a different travelling habits of people during Sundays as explained in Section III(a).

The average passenger wait time (AWT) for both non-peak and peak hours are captured in Fig 2.III(a)-(c) and Fig 2.III(d)-(f), respectively. We observe a small improvement in the AWT in case of taxi set 4 compared to the other taxi sets. However, this is not the case with peak hours, when the AWT value drops (smaller value preferred) for taxi sets 2, 3 which means the average waiting time is improved when taxis are injected to spots of low and high demands. This shows that additional taxis improves the chances of serving the incoming requests faster. However, we also notice a sharp increase in AWT when the taxi search radius is increased to 3 km. This means that the taxis at a further distance can serve the ride requests but will require additional time to reach the passenger.

The average number of messages (ANM) exchanged in the RSS ranges from 6 to 43 for non-peak hours as shown in Figures 2.IV(a)-(c). With an increase in the number of taxis from taxi set 1 to 4, we notice the ANM also increases. Further, the ANM for 3 km is more than that for 2 km radius for all the results. This is because
more taxis are available at a larger radius and hence more messages are sent back and forth to confirm the rides. From Figure 2.IV(d), it is seen that ANM during peak has much larger values as compared to non-peak hours, but even in this case the ANM value increases with more number of taxis as seen in taxi set 2, 3 and 4.

The average request processing time (ARPT) for non-peak hours as shown in Figure 2.V(a)-(c) and for peak hours in Figure 2.V(d)-(f). Smaller ARPT values are observed for non peak hours. The increase in value occurs both due to additional taxis as well as increase in the taxi search radius. An almost identical trend is observed for peak hours.

D. Performance Comparison with Benchmark Results

We have compared the results obtained using our method to the results obtained using the Taxi Ride Sharing (TRS) algorithm [5] explained in Section II. Their scheme planned to handle incoming requests sequentially, thus the request processing time was longer,
and the overall performance worsened. In contrast, our method handles requests in parallel, which helps to reduce the waiting time and improves the performance of the ride-sharing system. The comparison results are shown in Table IV.

We can see that the overall success rate of ride-matching (OSR) is 91.74% for non-peak hours compared to TRS which is 32.4%. For peak hours we see a lower OSR of 63.29%, but it is much higher than the TSR having 19.7%. The SRNP determines how well the ride requests are handled by matching them as ride-shares and by channeling incoming ride requests to taxis that are already matched with passengers sharing similar routes. We have collected data for taxis that are matched with 4, 3, 2 and 1 passenger. Our goal is to maximize the ride-share by having higher numbers for SNRP with more than 1 passenger. From the table, we can see a higher SNRP is achieved for both peak and non-peak hours using RSS with PSO. Similarly, for other metrics also, ridesharing using PSO is more efficient than the TRS algorithm.

VI. CONCLUSION AND FUTURE WORK

In this paper, we proposed a dynamic and hybrid model using a Particle Swarm Optimization based evolutionary (heuristic) algorithm for the ride-sharing system (RSS). Extensive experimentation using large-scale taxi ride data from Chicago shows that our algorithm can provide ride-sharing services to several passengers and simultaneously minimize the distance traveled by the taxis, thereby reducing the commute time and the carbon footprint. This algorithm can be applicable to other large scale data from other cities such as Chicago, New York and San Francisco as well. In future, we plan to extend our algorithm to multi-objective optimization taking into account the revenue of the taxi drivers. We shall also propose an efficient revenue model.

TABLE IV: Comparison of RSS using PSO and TRS Algorithm

| Parameters                        | RSS using PSO | TRS Algorithm |
|-----------------------------------|---------------|---------------|
|                                  | Non-Peak Hour | Peak Hour     |
| No of Requests                    | 121           | 1896          |
| No of Taxis (Set 1)               | 37            | 490           |
| Overall Success Rate of Ride Matching (OSR) % | 91.74        | 63.29         |
| Success Rate Nth Passengers (N=4) % | 23.42        | 28            |
| Success Rate Nth Passengers (N=3) % | 23.42        | 28.42         |
| Success Rate Nth Passengers (N=2) % | 25.23        | 28.42         |
| Success Rate Nth Passengers (N=1) % | 27.93        | 15.17         |
| Distance Ratio (DR)               | 1.87          | 1.73          |
| Average Waiting Time (AWT) in secs| 17.14         | 120.13        |
| Average Number of Messages (ANM)  | 5             | 269           |
| Average Request Processing Time (ARPT) in millisecs | 13.81 | 361.91 |

REFERENCES

[1] FHWA. National Household Travel Survey, 2017. Accessed on 06.06.2018.
[2] Chicago Data Portal. Chicago data portal. https://data.cityofchicago.org/, 2017.
[3] City of New York. Nyc taxi limousine commission. https://www1.nyc.gov/site/tlc/about/tlc-trip-record-data.page, 2019.
[4] City of San Francisco. Datasf. https://datasf.org/opendata/, 2019.
[5] Kanika Bathla, Vaskar Raychoudhury, Divya Saxena, and Ajay D Kshemkalyani. Real-time distributed taxi ride sharing. In 2018 21st International Conference on Intelligent Transportation Systems (ITSC), pages 2044–2051. IEEE, 2018.
[6] Pedro M d’Orey, Ricardo Fernandes, and Michel Ferreira. Reducing the environmental impact of taxi operation: the taxi-sharing use case. In 2012 12th International Conference on ITS Telecommunications, pages 319–323. IEEE, 2012.
[7] Jeffery B Greenblatt and Susan Shaheen. Automated vehicles, on-demand mobility, and environmental impacts. Current sustainable/renewable energy reports, 2(3):74–81, 2015.
[8] Harilaos N Psaraftis. A dynamic programming solution to the single vehicle many-to-many immediate request dial-a-ride problem. Transportation Science, 14(2):130–154, 1980.
[9] Shrawani Silwal, Md Osman Gani, and Vaskar Raychoudhury. A survey of taxi ride sharing system architectures. In 2019 IEEE International Conference on Smart Computing (SMARTCOMP), pages 144–149. IEEE, 2019.
[10] Niels Agatz, Alan Erera, Martin Savelbergh, and Xing Wang. Optimization for dynamic ride-sharing: A review. European Journal of Operational Research, 223(2):295–303, 2012.
[11] Pedro M d’Orey, Ricardo Fernandes, and Michel Ferreira. Empirical evaluation of a dynamic and distributed taxi-sharing system. In 2012 15th International IEEE Conference on Intelligent Transportation Systems, pages 140–146. IEEE, 2012.
[12] Mustafa Lokhandwala and Hua Cai. Dynamic ride sharing using traditional taxis and shared autonomous taxis: A case study of nyc. Transportation Research Part C: Emerging Technologies, 97:45–60, 2018.
[13] Fu-Shiung Hsieh, Fu-Min Zhan, and Yi-Hong Guo. A solution methodology for carpooling systems based on double auctions and cooperative coevolutionary particle swarms. Applied Intelligence, 49(2):741–763, 2019.
[14] Google. Google. https://cloud.google.com/maps-platform/, 2017.
[15] HERE. Here. https://developer.here.com, 2019.
[16] Inc Postman. Postman. https://www.getpostman.com/, 2019.
[17] Shrawani Silwal. A dynamic taxi ride sharing system using particle swarm optimization, ms thesis. https://drive.google.com/file/d/1wsusCTExXP9ancG1J2d_HAYWX1dDQ/view?usp=sharing, 2020.