1st International Conference on Applied Soft Computing Techniques
22 & 23.04.2017
In association with
International Journal of Scientific Research in Science and Technology

Data Driven Simulation Framework for Taxi Ride Sharing

P. Potri Rathna[1], Dr.T.Revathi[2]
[1]*Information Technology, Mepco Schlenk Engineering College, Sivakasi, Tamil Nadu, India*
rathnaprabhakaran@gmail.com[1]
[2]*Head of Department, Information Technology, Schlenk Engineering College, Sivakasi, Tamil Nadu, India*
trevathi@mepcoeng.ac.in [2]

ABSTRACT

In the modern era vehicles are increasing exponentially with respect to its population. The urban cities are facing many challenges in transportation and energy consumption. The foremost approach will be taxi ride-sharing which effectively reduces traffic congestion, gasoline consumption, and pollution. Our proposed method will simulate a real-time data-driven framework for analysing the taxi ride-sharing in various scenarios. In this approaches the taxies and trips are modelled as separate entities for simulating a rich set of realistic scenarios. A new optimization algorithm is described to address the computational complexity and scalability is achieved by an efficient indexing scheme combined with parallelization. The framework is evaluated using a real-time streaming information obtained from the user.

Keywords : Taxi Ride-Sharing, Shortest-Path, Scheduler, Scalability, Apache Spark

I. INTRODUCTION

Metropolitan cities are facing huge challenges due to a substantial increase in vehicles over the year, and this leads to traffic congestion, gasoline consumption, and pollution. Optimal strategy to decrease the stream of traffic and resource consumption [1] will be a taxi-ride sharing and at the same time, it needs to serve the transportation of city dwellers. The unused taxi can be efficiently filled by ride-sharing services. Every country wants to minimize its traffic and pollution, and the taxi company wants to get higher profit in each margin; and people wants to reach their destination faster with minimal cost. Sharing a taxi-ride has been identified as an optimization problem where the aim is to identify an optimal ride-sharing schedules [2], [3], [4], [5], [6], [7]. Some private organization is already providing ride-sharing services such as Uber, Ola, Lift, Via, Bandwagon and Cab With Me.

The process of deploying taxi-ride sharing needs a better understanding of its tradeoffs. The data-driven approaches are applied by providing a large volume of data for better understanding of the problem. The tradeoffs of taxi sharing are very challenging because there are multiple stakeholders with different and often with conflicting interests. Initially, the problem has been devised on the basis of survey data and analysis of psychological incentives. The graph-based model to this problem has been proposed by Santi et al. [8] which computes optimal sharing strategies for the trips and it uses two key parameters namely: maximum number of trips that can be shared and the maximum delay the passengers are willing to tolerate. This helps to study the passenger's inconvenience in sharing a taxi and for processing this model the passengers trips details need to be known in advance.

Our proposed simulation framework helps in analysing different ride-sharing scenarios. In this model, the passenger's trips are need not be known in advance and it fits well with models using different vendors. The proposed method helps in studying the realistic scenarios by providing a rich set of variables and by modelling taxi and trips as different entities which consider the different constraint of multiple stakeholders. The model includes the various variables like a maximum number of additional stops and waiting time,

and taxi-specific constraint like a number of passengers per taxi, a maximum number of shared trips. A new linear optimization algorithm is proposed for efficient indexing and assigning a trip to the taxis based on the cost factor. The efficiency of the method is analysed by providing a real-time streaming information.

II. RELATED WORK

The dynamic pickup and delivery problem [2], [3], has been addressed by linear programming [4], [5], [6]. This will be suitable for small-scale problems, for example, sharing within a certain distance. The real-time taxi dispatching uses heuristic-based method [12] but the scalability is very limited.

The data-driven method evolves with larger benefit and flexibility and the proposed method is similar to Santi et al. [8]. They used a "shareability network", where a node represents taxi trips, and the nodes get connected if they share the trips. This shareability network depends on two parameters the maximum number of shared trips per service and the maximum delay a passenger can tolerate in sharing a taxi. Let k be the maximum trip it can share and Δ be the maximum delay. This problem may lead to NP-hard for the higher k value and, it is tractable only for k=2. Similarly, the network size increases with Δ value, for larger Δ value the network will be larger and this will increase the computation time. However, the obtained results from the model will not be feasible for real time scenarios because this will not explicitly consider the taxi positions and their capacity

For example, there are two trips t_1, t_2 and this trips needs to be shared in a beneficial manner. For serving the passenger request there should be a cab c. Consider that c is serving some trips t_1 and some other trip t_i. If t_1 and t_3 are served by c in this order then it can't serve to the request of t_2 and similarly, if t_1 and t_2 are assigned to c then it can't serve to t_3 anymore or if it serves then the cost will be more.

According to this method, the trips of the passenger has to be known in advance for processing, this will be another limitation of this. This method is well suited for car pooling because in car pooling the time and location of each trip are fixed but this assumption is not suitable for taxi ride-sharing where trip request arrives at dynamic and real. To solve this issue Santi et al. proposed a refinement technique to their model which prunes the shareability network by keeping a time window δ. However, in real time, this works only for δ=0.

Ma et al. [9], [10] proposed a taxi ride-sharing dispatching method in real-time. Ma et al. splits the region into grid cells and calculate the distance "heuristically". This attains the fastest response because the heuristic calculation provides the shortest path with minimal computation. The accuracy of the system is very less and the results depend on the selected grid size. Similar to Ma et al. the proposed method response time will also be fast if the queries match with the cabs. Cache coherent indexing scheme is used for finding exact shortest path scheme for optimization.

Hung et al. [11] scheduling algorithms match the taxi with dynamic passenger trip requests. The taxis are scheduled with minimal cost by satisfying trip waiting and service time constraint. This uses Kinetic tree algorithms like branch-and-bound and mixed-integer programming. This type of kinetic algorithms can be integrated with the proposed algorithm.

III. DATA DRIVEN SIMULATION

When a dynamic trip request is issued by the passenger, our model schedule a taxi for the trip request by optimizing the pre-defined cost function under the constraint.

3.1 Simulation Components

Taxi Fleet: Indicates the set of taxis that are participating in taxi ride-sharing and the taxis are on dynamic. Here taxis are considered as a distinct entity which includes parameters like a maximum number of shared trips, passenger capacity, maximum pickup waiting time, extra time for drop off, vehicle speed, occupancy of people, the list of stops for sharing.

Passengers: People involve in sharing and the passenger size should be more than or equal to one. Includes the parameters like drop off location and a set of sharing constraint.

Scheduler: The scheduler schedules the appropriate taxi for the request based on ride sharing constraint. The scheduler must aware of taxis location and occupancy status.

Road Network: A directed graph G(V,E) is used to represent the road network. when a vehicle flows in both direction of a road then the bi-directed graph is used. Weights are used to incorporate the traffic condition.

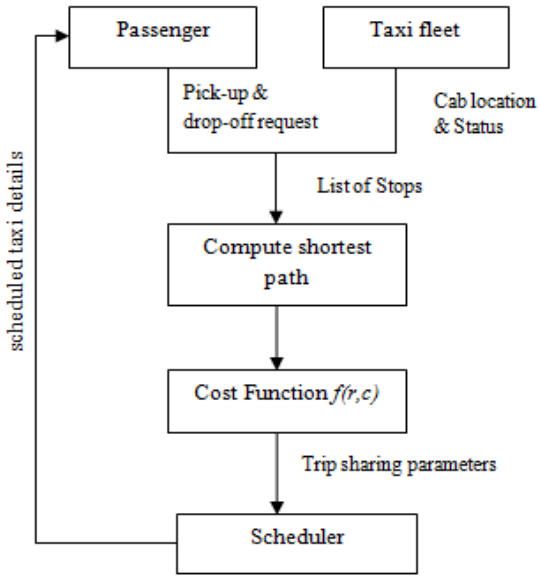

Figure 1: Overview of the Taxi Ride-Sharing

3.2 Data-Driven Simulation

The ride-sharing scenarios are simulated based on a set of input parameters and operate in an event-driven fashion where the states are updated when a new pickup request has been issued. When a new pickup request issued by the passenger then all taxis needs to update their status to the scheduler. Then the scheduler picks an appropriate taxi with minimal cost by computing the additional cost function *f*.

Input Parameters	Description
m	number of taxies involving in ride-sharing
C	Default taxi capacity
n_{share}	Maximum number f trips to be shared
t_{delay} /d_{delay}	Additional time/distance delay each customer could spend by default
t_{extra} / d_{extra}	Extra time /distance each customer could spend by default
f(r,c)	A cost function that a given taxi *C* and a pick-up request *r*, returns the cost of accommodating r with C.

Table 1: Input parameters for simulation

IV. SIMULATION ALGORITHM

Figure 1 shows the architecture of the proposed method. The aim is to minimize the total cost or maximize the total utility while meeting a set constraint. Let the additional cost function be $f(r_i, c_j)$ for a cab c_j for the trip r_i, n be the number of trips and m be the number of taxis. The total travel cost T(i) for the initial i trips can be calculated using

$$T(i) = \begin{cases} T(i-1) + min_{1 \le m}\{f(ri, cj)\} \\ T(i-1), if\ there\ is\ no\ available\ cabs \end{cases}$$

All the trips are considered in a chronological order. The real time dispatching is done by minimizing *Tc(n)*. The cab detail c_j is updated for every trip t_i based on the elapsed time.

4.1 Simulation Phase

The simulation takes the data in an event-driven fashion for scheduling the trip with a taxi of minimal cost with respect to set of constraint. The parameters are shown in table 1.

Algorithm 1 Simulating taxi ride-share

Input: a set of trips, R={r1,r2}

Parameters : m-no.of c abs, c-capacities of cabs, n_{share}, n_{delay}, n_{extra}, f(r,c)

STEPS :

1.Sort the values of R in chronological order and store it in R
2. Initialize the capacity to all cabs C {c1,c2,…., c_m }
3. Iterate i till |R| by assigning i value as 1
4. Subtract TimeofPickUp(r_{i-1}) from TimeofPickup(r_i) and store it in ElapsedTime
5. Assign infinity(∞) to f*
6. Iterate j till m by assigning j value as 1
7. updatecabstatus (c_i)
8. Assign f($r_{i,cj}$) to f_{ij}
9. if f* greater than f_{ij} then
10. assign f_{ij} to f*
11. assign c_j to c*
12. end if
13. end for
14. Assign trip r_i to the cab c* (r_i ,c*)
15. Add T(i-1) and f* and store the value in T(i)
16. end for

As of Algorithm 1, each trip is considered in chronological order. For each passenger pickup request t_i, examine the cab C status and update the status based on their elapsed time. Additional cost $f(t_i,c_j)$ for each cab is calculated and assign ri to the cab with minimal additional cost. The speed of the cab is cab is needed for updating its state hence, calculate speed using the trip duration and distance.

Simultaneously, update the cab occupancy and planned stops.

Algorithm 2 Cost Function *f(t,c)*

Input: r: trips, c:cab

Parameters : n_{share} ,n_{delay} ,$n_{extra,}$

STEPS :

1. Assign list of stops of C including its current location $\{s_0, s_1, \ldots, s_k\}$ to S
2. Calculate Shortest Path between s_k and $r.p_{drop}$ and store the value in D_{drop}
3. Compute FindPickUpLocation(r,c,s , D_{drop}) and store the value in idx_{pick} ,D*
4. Check if idx_{pick} is less than or equal to k then
5. subtract D_{drop} from D* and store in D_{pick}
6. Compute FindDropOffLocation (r,c,s ,idx_{pick} ,D*,D_{pick}) and store the value in idx_{drop} ,D*
7. end if
8. return D*

Output D* : an additional distance for cab c to accommodate

The Additional cost calculated in Algorithm 2, is used for finding the optimal route for the cab c_j which includes the pick-up and drop-off locations of r_i and compare its cost with the current route for c_j. This optimal route computation is called as Sequential Ordering Problem (SOP) which is a kind of Travelling Salesman Problem [18]. The heuristic search is used for computing the best route for c_j, for this first find the pick-up location p_{pick} and place it in the stop S and assume that the order to visit the stops are same and similarly, the drop-off p_{drop} is added to end of the route. While computing additional cost $f(r_i,c_j)$, the occupancy of the taxi is also checked. Once r_i to the cab c_j is assigned with the minimal cost, update the stops S details . The list of scheduled stops of cab c_j will be $S=\{s_0,s_1,\ldots\ldots s_k\}$, and p_{pick} and p_{drop} be the pick-up and drop-off locations of r_i.

Let us assume that the drop-off will happen after the last stop s_k for computing the additional distance to accommodate r_i Insert p_{pick} between s_{l-1} and s_l. If D_1, D_2 and D_3 be the lengths of shortest paths between $s_{l-1}$and p_{pick}, p_{pick} and s_l, and s_{l-1} and s_l then the additional distance will be defined as, $D= D_1 + D_2 – D_3 + D_{drop}$, and the algorithm performs pruning and stops if the delay constraint is no longer satisfied (Algorithm 3) by using this algorithm pick-up order of passenger is calculated.

Algorithm 3 FindPickUpLocation

Input: r trip, c cabs, $s=\{s_0, s_1, \ldots, s_k\}$ list of stops of c including its current location, D_{drop} the shortest distance between s_k and p_{drop}

Fields: c.C capacity of c, $r.o_p$ number of passengers of r

Parameters: n_{share}, n_{delay}, n_{extra}

STEPS:

1. Assign the cab passenger count value to idx
2. Assign infinity(∞) to newdist(D*)
3. Increment the k value and store it in idx_{pick}
4. Iterate the for loop till k, by assigning idx+1 value to j.
5. Compute the shortest path between previous location and current pick-up location then store the value in D_1
6. Compute the shortest path between current pick-up location and $stop_j$ location then store the value in D_2.
7. Compute the shortest path between previous location and $stop_j$ location then store the location in D_3.
8. If d_{delay} <= 1 && d_{extra} <= 2 for all trip then
9. $D=D_1 + D_2 - D_3 + D_{drop}$
10. Check if olddist(D) < newdist(D*) then
11. Assign D value to D*
12. Assign j value to idx_{pick}
13. end if
14. else if d_{delay} not satisfied for trip r
15. break
16. end if
17. assign s_j value to prev
18. end for

Return idx_{pick}. D*

Similarly, the drop-off of the passenger is computed using the Algorithm 4. In the algorithm, the best position for p_{drop} is searched. As like the FindPickUpLocation process, for each $s_l \epsilon \{s_{p-1} = p_{pick}, s_p, s_{p+1}, \ldots\ldots s_{k-1}\}$. Then query the shortest path between s_l and p_{drop}, p_{drop} and s_{l+1}, and s_l and s_{l+1}, will be D_4, D_5, D_6 respectively. Then the additional cost of each new route will be defined as , $D = D_{pick} + D_4 + D_5 - D_6$. For example, consider $l=5000$, $M=30$ and $\epsilon = 0.2$, the probability that the minimal cost derived by the heuristic would be more than 0.2.

Algorithm 4 FindDropOffLocation

Input: r trip, c cabs, s={s_0, s_1,….., s_k} list of stops of c, idx_{pick} index of S where to insert p_{pick}, D* additional distance obtained from FindPickUpLocation, D_{pick} additional distance to insert p_{pick} at idx_{pick}
Parameters : d_{delay}, d_{extra}
STEPS:
1. Assign p_{pick} value to prev
2. Iterate the for loop till k, by assigning idx_{pick} value to j
3. Compute the shortest path between the previous location and passenger drop location then store the value in D_4
4. Compute the shortest path between passenger drop location and stop(s_j) location then store the value in D_5.
5. Compute the shortest path between previous location and stop(s_j) location then store the location in D_6.
6. Check if d_{delay} <= 1 && d_{extra} <= 2 for trips are satisfied then
7. $D=D_{pick} + D_4 + D_5 - D_6$
8. If olddist(D) is less than newdist(D*) then
9. Assign D value to D*
10. Assign j value to idx_{drop}
11. end if
12. end if
13. assign s_j value to prev
14. end for

Return idx_{drop}. D*

4.2 Cache Coherent Shortest Path Index

The shortest path queries are used extensively in our algorithm (Algorithm 2, 3, 4), at this point the computation spends much time. Each computation of $f(r_i,c_j)$ makes a series of the queries for all stops of c_j, for the finding solution with minimal cost. Initially, precompute all the shortest distance between each nodes and cache the distance for all intersection pairs. This will reduce the cache misses and can be used for fast retrieval of distance if the same queries repeated. The matrix storage size is simply small and would fit completely on commodity computers. Thus the shortest path queries are now reduced to memory access.

An efficient and easy method to increase cache coherent of shortest path lookups has been proposed. Transport the shortest path matrix, forward lookups become backward lookups and vice versa and an additional transposed matrix has been included in the shortest path queries to convert all forward lookups to backward lookups. The cache-coherent layout will systematically reduce the number of cache misses. This efficiently reduces up to six times than the single core.

V. EXPERIMENTAL SETUP

The experiment is performed with real-time user data. For getting the user information the android application is used and for executing the process Apache Spark is used. The streamed dynamic data is processed continuously using Spark.

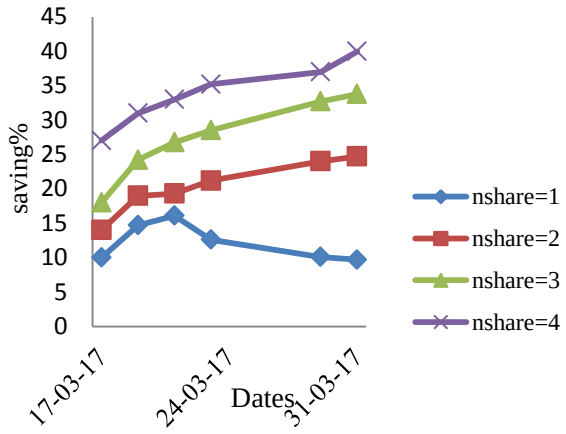

Figure 2. The saving percentage of total cost for travel distance through sharing ride for n_{share}=1,2,3,4.

5.1 Data

An Android application is created for getting user details which include information like a passenger name, number of passengers for a ride, pick-up and drop-off location and their date and time of travel. The application is created using Android Studio IDE. The application will use the Google Map detail for getting the user pick-up and drop-off location. In this, the android application will act as a client end which sends data to the server.

The Apache Spark will act as a back end for processing. It seamlessly gets user data dynamically and runs the scheduling algorithm for scheduling the user request with an appropriate cab. For processing the user request each trip is represented with the following fields: taxi ID, pick-up time, pick-up and drop-off locations, travel distance in kilometers and number of passengers details are stored in server side. Assume that the maximum number of extra stops, which is at most $2n_{share}$. For keeping the waiting and service times as reasonable, set

d_{delay}=5 minutes and d_{extra}= 10 minutes and set each taxi's capacity C=4 for simplicity. The Fig.2 explains the total cost saving % for a passengers who undergone for sharing taxis where n_{share}=1,2,3,4

VI. CONCLUSION

In this paper, we present a data-driven simulation for taxi-ride sharing to improve flexibility and scalability for rich set of realistic ride-sharing scenarios. The main goal of scheduling algorithm is to work seamlessly in allocating a taxi request with a cab of minimal cost Cache-coherent layout helps to speed up the shortest path queries and speed up the entire processing. The implementation of our model if fully done with Apache Spark, which enables of variety batch analysis tasks. The future work we would like to implement the load balancer for the shortest path queries. In this, the shortest path database could be loaded on a separate machine. This will allow making a better use of computing resources when having multiple simulator instances.

VII. REFERENCES

[1]. V. Handke and H. Jonuschat, Flexible Ridesharing. Springer Berlin Heidelberg, 2013.

[2]. J. Yang, P. Jaillet, and H. Mahmassani, "Real-time multivehicle truckload pickup and delivery problems," Transportation Science, vol. 38, no. 2, pp. 135–148, 2004.

[3]. G. Berbeglia, J.-F. Cordeau, and G. Laporte, "Dynamic pickup and delivery problems," EJOR, vol. 202, no. 1, pp. 8 – 15, 2010.

[4]. A. Marin, "Airport management: taxi planning," Annals of Operations Research, vol. 143, no. 1, pp. 191–202, 2006.

[5]. G. Keith, A. Richards, and S. Sharma, "Optimization of taxiway routing and runway scheduling," in AIAA Guidance, Navigation and Control Conference and Exhibit, 2008.

[6]. P. Shaw, "Using constraint programming and local search methods to solve vehicle routing problems," in Principles and Practice of Constraint Programming CP98, ser. Lecture Notes in Computer Science, M. Maher and J.-F. Puget, Eds. Springer Berlin Heidelberg, 1998, vol. 1520, pp. 417–431.

[7]. B. Coltin and M. Veloso, "Scheduling for transfers in pickup and delivery problems with very large neighborhood search," in AAAI, 2014, pp. 2250–2256.

[8]. P. Santi, G. Resta, M. Szell, S. Sobolevsky, S. Strogatz, and C. Ratti, "Quantifying the benefits of vehicle pooling with shareability networks," PNAS, vol. 111, no. 37, pp. 13 290–13 294, 2014.

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

[10]. S. Ma, O. Wolfson, and Y. Zheng, "Real-time city-scale taxi ridesharing," IEEE Transactions on Knowledge Discovery and Data Engineering, vol. 27, pp. 1782–1795, 2015.

[11]. Y. Huang, F. Bastani, R. Jin, and X. S. Wang, "Large scale real-time ridesharing with service guarantee on road networks," PVLDB, vol. 7, no. 14, pp. 2017–2028, 2014.

[12]. W. M. Herbawi and M. Weber, "A genetic and insertion heuristic algorithm for solving the dynamic ridematching problem with time windows," in GECCO '12, 2012, pp. 385–392.