An Efficient Solution for Ridesharing problem

Kopalle Himanish*, Krishna Chaitanya†, Maral Azizi‡
Department of Computer Science and Engineering, University of North Texas
Email: *himanishk opalle@my.unt.edu, †krishnasanagavarapu@my.unt.edu, ‡maralazizi@my.unt.edu

***Abstract*—Getting a taxi in highly congested areas (e.g. airports, train stations) is both time consuming and expensive. While the emergence of novel Transportation Network Companies (e.g. Uber) has helped increase the supply of drivers during peak times, they have done little to reduce congestion in hot spots such as airports, major stations, and stadiums. In these places, a steady stream of passengers arrives via some public transport mode, say train, and then depart to different destinations. The large volume and continuous flow of arriving passengers create a golden opportunity for ridesharing among departing passengers. We propose a ride sharing algorithm, which will allow a selective passenger whose source to destination path shares a common path. This algorithm will not only provide the ride to share among the passengers but also allow the passengers cost to reduce making the ride cost effective. A ride sharing can be significant social as well as environmental benefit by saving the energy consumption.**

Keywords: GIS, ride sharing, pg routing, Dijkstras, spatial database, shortest path, OpenStreetMap, PostgreSQL, QGIS, TSP

I. Introduction

As we know taxi is a very popular transportation facility because of its convenient and its availability most of the passengers prefer to use taxi instead of other public transportation. In the other hand, number of passengers is more than number of taxies especially in metropolitans like New York City people mostly have to take a taxi because of traffics issue and non-availability of parking places. However, taxi is more expensive compare to other type of transportation like bus, train or even airplane. Taxi expensiveness is due to this fact Taxi is a private transportation. The amount of gas and mileage that is being used by a taxi must be paid by passenger, while, it can be share between number of passengers. To address this issue there are two simple ways, first one is increasing the number of taxies, and second is sharing taxi among passengers. Having more taxies will cause more traffic and gas consumption [9]. Specially after oil crisis in 1973 ride-sharing became a serious industry in the US [2]. Although we are not in the oil crises ages now but still energy crisis is any significant in the supply of energy resources to an economy. Industrial development and population growth have led to a surge in the global demand for energy in recent years.

However, ridesharing is not an easy task, it needs accurate prediction of mileage and road metrics such as passengers destination, common paths, etc. This problem made us think to create a ride sharing algorithm for real world ride sharing issue. To perform this task, we decided to examine applicability of our approach by Uber. Uber is an American multinational online transportation network company headquartered in San Francisco, California. It develops, markets and operates the Uber mobile app, which allows consumers with smartphones to submit a trip request which is then routed to Uber drivers who use their own cars.

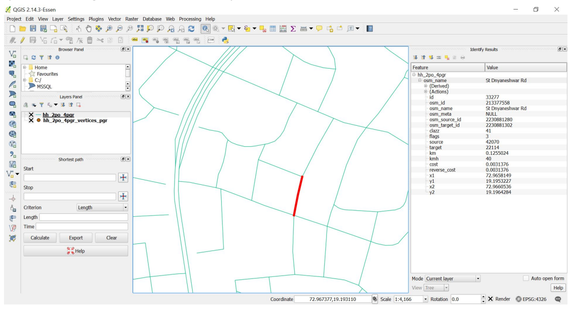
Although ride-sharing is not a new definition, it is a solution for saving energy consumption and assuaging traffic congestion while satisfying peoples needs in commute [10]. Ridesharing based on private cars, often known as carpooling or recurring ridesharing, has been studied for years to deal with peoples routine commutes, e.g., from home to work.

In this study, we are developing an algorithm for ride-sharing that can be use by Uber application. Our suggested algorithm gets two passengers information, e.g., source, destination and it will find the shortest path from source to destination and also, it will find a common path between two passengers which is study the taxi ridesharing problem in a practical setting. The rest of this paper is organized as follows. In Section II, we formally define the dynamic taxi ridesharing problem and overview our proposed service. Section IV introduces the we will describe background of this study. Section II describes our experimental set up. We present the evaluation in Section III and conclusion in Section V.

II. Methodology

A. Experimental Setup

In this project, we have used OpenStreetMap data, built by a community of mappers that contribute and maintain data about roads, trails, cafes, railway stations, and much more, all over the world. The OpenStreetMap License allows free (or almost free) access to our map images and all of our underlying map data. It is a collaborative project to create a free editable map of the world. In this project, we have extracted the OSM data [7] from Geofabrik data extracts. The dataset for the experiment is OSM data which is a .pbf file containing the roads of the selected part of the world. Setup the osm2pgsql on windows after downloading the OSM data. osm2pgsql is a command-line based program that converts OpenStreetMap data to postGIS-enabled PostgreSQL databases. Following is the mandatory steps to complete the setup, add the path of the osm2pgsql executable file to the system path in the systems environment variables. Executing osm2pgsql command in the windows command prompt which should result in showing version number. Run the osm2pgsql program via command line and import the data into the database created in PgAdmin III. The database for the current project in PgAdmin is India. Executing the following command pushes all the data into database. Importing the data loaded in the database with the help of QGIS (Quantum GIS) shows the routes by adding PostGIS layers.

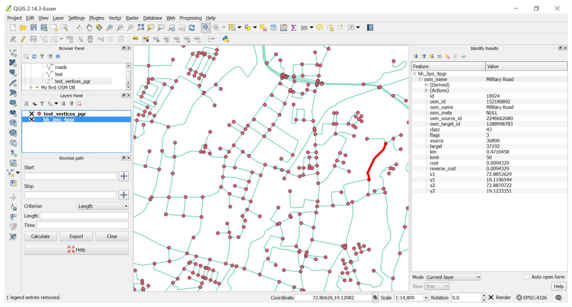


Feature	Value
hh_2po_4pgr	
osm_name	St Dnyaneshwar Rd
(Derived)	
(Actions)	
id	33277
osm_id	213377558
osm_name	St Dnyaneshwar Rd
osm_meta	NULL
osm_source_id	2230881280
osm_target_id	2230881302
clazz	41
flags	3
source	42070
target	22114
km	0.1255024
kmh	40
cost	0.0031376
reverse_cost	0.0031376
x1	72.9658149
y1	19.1953227
x2	72.9660536
y2	19.1964284

Fig. 1: Example of Imported OSM data set

OSM2PO is a converter and a routing engine. Its converter parses OpenStreetMap's XML-Data and makes it routable. It generates sql files for PostGIS, compatible with pgRouting and Quantum GIS. It creates compact topology/graph files for the integrated routing engine. The installation files include an osm2po executable jar file, a windows batch file, a shell file and some configuration files. Edit the windows batch file available after extracting all files and update the URL for OSM file. Execute the batch file in windows command prompt. Run the PostgreSQL file created from the above process which generates tables and schemas for the corresponding data. Run a PostgreSQL function `pgr_createVerticesTable` to create vertices for the dataset we have imported into QGIS.

In the ride sharing algorithm, we will be taking



Feature	Value
hh_2po_4pgr	
osm_name	Military Road
(Derived)	
(Actions)	
id	18924
osm_id	152180892
osm_name	Military Road
osm_meta	NULL
osm_source_id	2246662680
osm_target_id	1288998783
clazz	43
flags	3
source	36890
target	37192
km	0.4716458
kmh	50
cost	0.0094329
reverse_cost	0.0094329
x1	72.8853629
y1	19.1196544
x2	72.8870722
y2	19.1233351

Coordinate 72.86626,19.12982 Scale 1:14,809 Rotation 0.0 Render EPSG:4326

Fig. 2: Imported OSM data with vertices and roads

two sources by two different users and a common destination.

1) Constraints: The ridesharing problem is to dispatch taxis to ride requests, subject to certain constraints. Below we defined constraint for this study:

- In the algorithm, we take routes from different sources to single destination.
- Consider predefined cost per mile in order to calculate total and shared costs for the ride.

2) Algorithm: The pseudo code of our proposed algorithm is presented below:

```
Input <- Source S1, S2 and D.
Calculate shortest path from S1
to D and S2 to D is found using
pg_Dijkstras.
Calculate common point common in both
S1-D and S2-D.
If S1-D > S2-D
If common == S2
Final_path <- S1-D
Cost <- cost(S1,D)
Else
Final_path <- S1-D  Cost <- cost (S1,
S2,D)
Else
If common == S2
Final_path <- S1-D
Cost <- cost(S1,D)
Else
Final_Path <- S1-D
Cost <- cost (S1, S2,D)
```

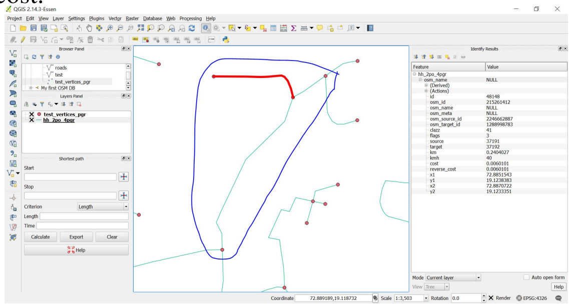
In the first step store the source and destination locations given as input from the user. Calculate the individual paths and distances from all the sources to the destination. Calculate the longest path in all the individual paths calculated. After getting the longest path `(S1->D / S2->D)`, we store all the vertices's between S1 to D and vertices between S2 to D. Then we calculate the common point common point. Here, common point is the first intersection point of the two paths `S1->D and S2->D`. Once we get the common point we find the common path. Common path is the distance between the common point and destination. If the common path is greater than the distance between source S1 to common point or source S2 to common point, then the final path will be the maximum of the S1 to D and S2 to D. Once we get the final path, we find the total cost which have been shared and the cost which have been reduced for both the users. Thus we get a final path which is not just shortest but also cost effective.

III. Data and Analysis

This section evaluates the effectiveness and efficiency of our proposed algorithm. We divide the problem into three sub problems.

Case 1:

Conditions: `S1->D > S2->D` There exists a common point between the intersection of `S1->D and S2->D` `Common(S1->D, S2->D) = c` Process: In this case, we get a common point c, which is the first intersection between the paths from S1 to D and S2 to D. Here, we calculate the distance from c to D and use the function check () to find the final path. We also calculate the cost from source S1 to D and source S2 to D, and determine the shared cost and the total saved cost for both the users. Here the shared cost is half the cost of the shared path. Since the shared path is the route from c to D. So the shared cost is half the cost of `c->D` path. The total cost from `S1->D` will be the sum of the cost of path from S1 to c and shared cost. Furthermore, the total cost from `S2->D` will be the sum of the cost of path from S2 to c and shared cost.



```
Identify Results
Feature              Value
hh_2po_4pgr
  osm_name           NULL
    (Derived)
    (Actions)
    id               48148
    osm_id           215261412
    osm_name         NULL
    osm_meta         NULL
    osm_source_id    2246662887
    osm_target_id    1288998783
    clazz            41
    flags            3
    source           37191
    target           37192
    km               0.2404027
    kmh              40
    cost             0.0060101
    reverse_cost     0.0060101
    x1               72.8851543
    y1               19.1238383
    x2               72.8870722
    y2               19.1233351
```

Fig. 3: Example route for case 1

Below in Table I - III we presented the values and parameters for each attribute in case 1.

TABLE I: Inputs value for ridesharing algorithm in Case 1

Case 1	Values
S1	35248
S2	35060
D	35059

TABLE II: - Results values of `pgr_Dijkestra` function for Source 1 to destination in Case 1

Seq Integer	id1 (point)	id2 (edge)	Distance (km)
0	35248	46118	0.262122
1	35247	45915	0.1905067
2	35059	-1	-

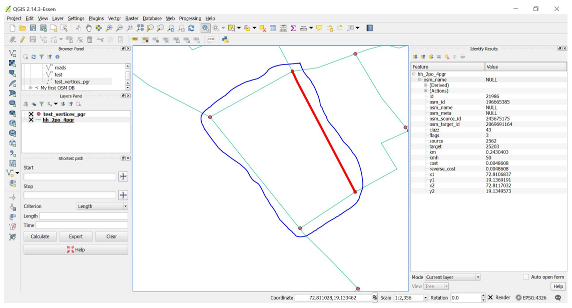
Case 2:

Conditions:

`S1->D > S2->D` Let S2 be the common point between `S1->D and S2->D Common(S1->D, S2->D) = S2` Process: In this case, we have no external common point. The shortest path from S1 to D will pass through S2 to D. This makes the common point or first interaction point from `S1->D and S2->D` to be S2. The shared cost will be half the cost of the shared path. Here, the shared path is `S2->D`; which makes the shared cost to be half the cost from `S2->D`. So the final path will be from `S1->D` and the shared path will be `S2->D.` The total cost for user at S1 will be the sum of cost from S1 to S2 and shared cost. Similarly, the total cost for the user at S2 will be the shared cost from S2 to D.

TABLE III: - Results values of `pgr_Dijkestra` function for Source 2 to destination in Case 1

Seq Integer	id1 (point)	id2 (edge)	Distance (km)
0	35060	45916	0.262122
1	35247	45915	0.2495008
2	35059	-1	0



```
Identify Results
Feature              Value
hh_2po_4pgr
  osm_name           NULL
    (Derived)
    (Actions)
    id               21986
    osm_id           196665385
    osm_name         NULL
    osm_meta         NULL
    osm_source_id    245675175
    osm_target_id    2069691164
    clazz            43
    flags            3
    source           2562
    target           25203
    km               0.2430403
    kmh              50
    cost             0.0048608
    reverse_cost     0.0048608
    x1               72.8106837
    y1               19.1369191
    x2               72.8117032
    y2               19.1349573
```

Fig. 4: Example route for case 2

Table IV - VI shows the values and parameters for each attribute in case 2.

TABLE IV: Inputs value for ridesharing algorithm in Case 2

Case 2	Values
S1	24685
S2	24802
D	24683

Case 3:

Conditions: `S1->D <> S2->D` There exists no common point between the shortest paths of `S1->D`

TABLE V: Results values of `pgr_Dijkestra` function for Source 1 to destination in Case 2

Seq Integer	id1 (point)	id2 (edge)	Distance (km)
0	24685	35993	0.0737169
1	24686	36207	0.180339
2	24802	36121	0.2127011
3	24743	36120	0.221068
4	24683	-1	-

TABLE VI: Results values of `pgr_Dijkestra` function for Source 2 to destination in Case 2

Seq Integer	id1 (point)	id2 (edge)	Distance (km)
0	24802	36121	0.180339
1	24743	36120	0.2127011
2	24683	-1	-

`and S2->D`; which makes the common point to be destination point, D itself. `Common(S1->D, S2->D) = D`. Process: In this case, we have the common point as destination. The shortest path from `S1->D` will have a different path from the shortest path from `S2->D`. So, here we consider the cost from `S1->D`, say cost1 and cost from `S2->D`, say cost2. Furthermore, we calculate the cost of the path from S1 to S2, say cost3. We determine the shortest path for the user riding from S1 to the destination D by checking the minimum cost like, `Min(Cost3 + shared cost(half of Cost2) and Cost1)`. If the Cost1 is more than the common point will be S2 and the final cost effective path will be `S1->D`, Else there will be no ride sharing between the two users.

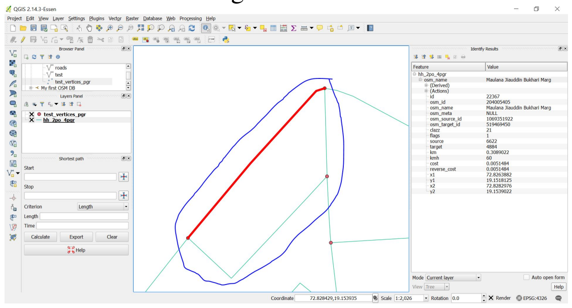


Fig. 5: Example route for case 3

Table VII - IX depict the values and for each attribute in case 3.

We have examined our proposed algorithm on 20 different datasets for different locations including long and short routes. Table X and XI shows the final results for case 1 and 2. As can be seen from these two tables our proposed algorithm is effective in terms of cost saving, we save 14 percent of total cost have been saved.

TABLE VII: Inputs value for ridesharing algorithm in Case 3

Case 3	Values
S1	22115
S2	22114
D	42070

TABLE VIII: Results values of `pgr_Dijkestra` function for Source 1 to destination in Case 3

Seq Integer	id1 (point)	id2 (edge)	Distance (km)
0	22115	33367	0.1317222
1	22135	53831	0.1363831
2	42070	-1	-

TABLE IX: Results values of `pgr_Dijkestra` function for Source 2 to destination in Case 3

Seq Integer	id1 (point)	id2 (edge)	Distance (km)
0	22114	33277	0.1255024
1	42070	-1	-

TABLE X: Results for Case 1

Attributes	Values
Final path	0.511623
shared distance	0.262122
final cost of ride	1.023246
Original cost for user 2 without ride sharing	1.023246
Cost for user 2 after ride sharing	0.761124
Saved cost for user 2	0.262122
Original cost for user 1 without ride sharing	0.905257
Cost for user 1 after ride sharing	0.262122
Saved cost for user 1	0.643135

TABLE XI: Results for Case 2

Attributes	Values
Final path	0.687825
shared distance	0.39304
final cost of ride	1.37565
Original cost for user 2 without ride sharing	0.78608
Cost for user 2 after ride sharing	0.19652
Saved cost for user 2	0.58956
Original cost for user 1 without ride sharing	1.37565
Cost for user 1 after ride sharing	0.491305
Saved cost for user 1	0.884345

IV. Related Work

In this section we will review some related work and study about ridesharing problem. By taking a look at the application stores like Android or iOS application stores you may find many ridesharing applications, moreover, there are bunch of ridesharing website over the internet which claim that their mobile app or website is highly effective for scheduling [6]. Although some of those apps and websites, have their uses and large user bases, the static routing problems they help solve makes those uses fairly limited.

Since our focus in this study about static ridesharing problem and not application development we will not dig into applications issues. Static ridesharing is a NP-hard problem itself as a result many researchers are focusing to provide a solution for this. By taking a look at related papers you may see almost all of the available papers and literature on carpool and ride-sharing mainly tackle the static ridesharing issues, whereby users must pre-schedule their trips, neglecting the dynamic aspect. However, there are some study in dynamic aspects of ridesharing [5], [3], [4], [1] but still static ridesharing is in its early stages and needs more research for optimization.

In [2] researches attempt to highlight the need for developing dynamic real-time carpool and ride-sharing solutions, instead of already outdated static ones, by applying some novel web technologies. Their main focus is creating an application to be able run in cloud for ridesharing. In some cases, researchers were chasing traceability and security in communication services, for example in [8] authors identified the security issues as one of the main reasons hindering their success.

Authors in [9] proposed and developed a mobile-cloud based real-time taxi-sharing system which is able to enhance the delivery capability of taxis in a city so as to satisfy the commute of more people and also the system saves the total travel distance of taxis when delivering passengers and moreover, the system can also save the taxi fare for each individual rider while the profit of taxi drivers does not decrease compared with the case where no taxi-sharing is conducted.

All cited current papers admittedly still provided us with a lot of beneficial ideas and food for thought transferred onto this paper, its findings and conclusions, and out of that still quite disorganized literature which tackles a lot of issues, we have identified some yet non-tackled, laid out in the following sections. For instance, in [10] authors proposed a practical large-scale taxi ridesharing service and they evaluated their service based on a GPS trajectory dataset generated by 33,000 taxis over 3 months, in which over 10 million queries were extracted. This paper was inspiring since their service can enhance the delivery capability of taxis in a city so as to satisfy the commute of more people. In our case, each taxi that is occupied under only for two passengers. Our work is also an example of urban computing, where a series of research work has been done with taxi directions.

V. Conclusions and Future Work

This paper presents a ride sharing algorithm, which will allow a selective passenger to share a ride whose source to destination path shares a common path. This algorithm will not only provide the ride to share among the passengers but also allow the passengers cost to reduce making the ride cost effective. A ride sharing can be significant social as well as environmental benefit by saving the energy consumption. Our suggested algorithm gets two passengers information, e.g., source, destination and it will find the shortest path from source to destination and also, it will find a common path between two passengers which is study the taxi ridesharing problem in a practical setting. This paper shows cost effective shared path and shared cost from both the sources to the destination. `pg_Dijkstras` algorithm is used to find the shortest path from source to destination. The novel approach contains path sharing and ride sharing techniques which will not only reduce the distance but also the cost. we have extracted the OSM data from Geofabrik data extracts to perform the experiments.

In future, we are trying to extend our algorithm which can handle more than one destination and more than two sources. Furthermore, we will incorporate willing to walk functionality in the algorithm in which user can save more money by walking to the destination from a point in case of multiple destinations. Adding more, an added functionality of rerouting in which a route can be chosen from different available paths from a source to destination in order to share a ride.

References

[1] A. Colorni and G. Righini. Modeling and optimizing dynamic dial-aride problems. In *International Transactions in Operational Research*, 2001.

[2] Vladimir Dimitrieski Dejan Dimitrijevi, Nemanja Nedi. Real-time carpooling and ride-sharing: Position paper on design concepts, distribution and cloud computing strategies. In *Computer Science and Information Systems*, page 781786, 2013.

[3] Y. C. Hsu S. N. Yang C. H. Gan J. L. Lu, M.Y. Yeh and M. S. Chen. Operating electric taxi fleets: A new dispatching strategy with charging plans. In *IEVC*, 2012.

[4] C. Zhang X. Xie J. Yuan, Y. Zheng and G. Sun. An interact voting based map matching algorithm. In *MDM*, 2010.

[5] A Lokketangen L. M. Hvattum and G Laporte. A branch-and-regret heuristic for stochastic and dynamic vehicle routing problems. In *Networks*, 2007.

[6] L. Zhang X. Xie N. J. Yuan, Y. Zheng. T-finder: A recommender system for finding passengers and vacant taxis. In *IEEE TKDE*, 2013.

[7] Manuel Pichler. PHP Depend - software metrics for PHP. [Accessed: Aug. 26, 2015].

[8] A distributed dijkstra's algorithm for the implementation of a real time carpooling service with an optimized aspect on siblings. In *In Intelligent Transportation Systems*, pages 795–800, July.

[9] Ouri Wolfson Shuo Ma, Yu Zheng. T-share: A large-scale dynamic taxi ridesharing service. In *ICDE Conference* , November 2013.

[10] Yu Zheng Shuo Ma. Real-time city-scale taxi ridesharing. In *IEEE TRANSACTIONS ON KNOWLEDGE AND DATA ENGINEERING*, 2015.

[11]