CS 581- SEARCH OF SPATIO-TEMPORAL RESOURCES:

Final Project Report

Abstract

The aim of these simulations is to compare and contrast different algorithms, with different levels of information access to find parking spots in real time. The setup is done for the city of San Francisco, CA. We use the data released by the parking authorities, ranging over a month, given with timestamps accurate to second granularity.

Group 3

Team Members:

Kruti Sharma

Ridhi Rustagi

Suvadeep Ghosh

Tanima Chatterjee

Acknowledgement

We would like to thank **Prof. Ouri Wolfson** for his pedagogical inputs and mentoring which enabled us to understand the intrinsic behavior of the various Spatio-temporal algorithms which we have applied in our project to deliver a successful product. His research experience in database management system domain proved fruitful and extensive when it came to understanding the project requirements and guiding us through the development lifecycle. We look at parking spot allocation with a completely different viewpoint after this project and appreciate the thought and execution that goes in it.

We would also like to thank the teaching assistant for this course **Mr. Sandeep Sasidharan** for his continued guidance and excellent feedback throughout the course of the project.

Contents

Sections Page No

ABSTRACT

The simulations in this project compares and evaluates various algorithms on a given Spatio-temporal dataset with different levels of information access to find parking spots in real time The algorithms try to deal with the parking issues faced by drivers to in big cities like San Francisco. The data used is the one released by the parking authorities, ranging over a month, given with timestamps accurate to second granularity.

The following approaches were discussed and implemented in the project:

1. Task 1 (DETERMINISTIC APPROACH)
   To evaluate different algorithms to find a parking spot for the user with high level of information on a real time dataset like parking block, parking availability, date and time, user destination and start location.

2. Task 2 (PROBABILISTIC APPROACH)
   To evaluate different algorithms to find a parking spot for a user with limited level of information like parking block, start location, destination location and time, and the dataset is probabilistic rather than real time.

3. Task 3 (BASELINE APPROACH)
   To find a parking spot for a user by performing an uninformed search having information about the user destination, starting location and parking block only.

Simulations are carried out for each implementation of each task with various each congestion levels and the corresponding results are plotted giving a pictorial representation of the performance of the algorithms in each of the above mentioned scenarios.

ALGORITHMS EVALUATED

Before we start explaining all the algorithms that we have evaluated, first we will describe how our algorithms behave as the user moves along the road network. This section is applicable to all the algorithms that we have evaluated.

Approximations

All our algorithms shall be invoked each time the user reaches a road intersection. We are using Google Maps API to get current location of the user as the user moves along the road network. Since the road intersection co-ordinates provided by the database did not match the road intersection co-ordinates provided by the API, we performed few approximations to identify the road intersection points and then trigger our algorithms.

Approximations to estimate the road intersection points

1. Initially we take all the intersection nodes from the database.
2. We take the midpoint of the parking blocks from the database and assume this as our approximated point where parking slot is located on the block.
3. We take the start location co-ordinates of the user and approximate it to the nearest intersection node by using the distance formula:

$$D = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

4. A point with minimum distance is given max weight and this point is made the approximated user starting location. This is done each time our algorithm is triggered at the intersection nodes.
5. Similarly, we take the user's destination location and approximate it to nearest parking block midpoint using the above distance function, as user can park at these points only.

Input and Output

Following is the set of inputs for each algorithm evaluated:

Input:

1. Start Location
2. End Location
3. Current Time

Following is the set of outputs from each algorithm evaluated:

Output:

1. Parking Slot Location
2. Total Time = Time required to acquire parking slot + walking time from parking slot to end location (for simulation analysis)
3. Query Time = Time taken by the algorithm to output the parking slot.

ALGORITHMS

We describe each algorithm with respect to the input and output sets and the specific steps involved. Evaluation procedure of algorithms and the underlying logic that each of these algorithms uses to find a parking spot is explained in detail below.

TASK 1: DETERMINISTIC APPROACH

Level of Information Available: Maximum

Maintenance Cost: Most Expensive

Accuracy: High

i. GRAVITATIONAL FORCE DISTANCE BASED ALGORITHM

The algorithm is based on the concept that the more the available parking spots a block has and the closer it is to the vehicle; the higher gravitational force it has to pull the vehicle towards it.

Procedure:

1. Get initial route from Google maps API using the input start and end location.

2. When the user clicks on "Navigate" button on GUI, get the current location of the user and check if those co-ordinates are intersection points by using the approximation described in the previous section (2.1).

3. If the co-ordinates are intersection points, then pass the current user location co-ordinates and current time to the SQL query.

4. The SQL query does the following:

    4.1 Calculates the maximum force towards each parking block using the following formula

    F = n (available)/(distance(current location, parking block)) 2
    Where:
    F = force
    N (available) = number of parking slots available in that block distance (current location, parking block) = distance of parking block from current user location

    4.2 Returns the co-ordinates of the parking block with maximum force

5. The co-ordinates returned by the SQL query are passed to another SQL query which gets the

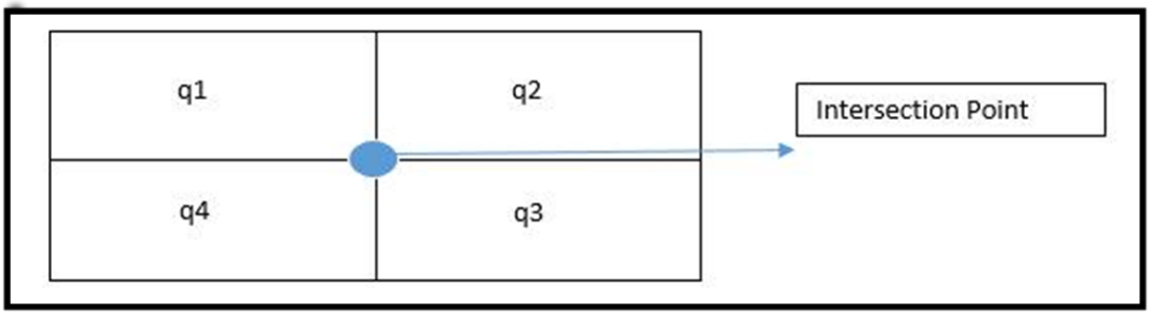
walking distance from the allocated parking block to the End Location.

ii. GRAVITATIONAL FORCE COST BASED ALGORITHM

In this approach, at each node, the parking block availability is divided with respect to four quadrants as in case of a 2-D graph.

Procedure:

1. Using the Start Location and the End Location get the initial route from the Google Maps API.
2. When the user clicks on "Navigate" button on GUI, get the current location of the user and check if those co-ordinates are intersection points by using the approximation described in the section 2.1.
3. If the co-ordinates are intersection points, then use the current user location and current time to form 4 quadrants as shown below:



4. The quadrant with maximum availability is considered.
5. Pass the co-ordinates of the parking block with maximum availability to the SQL query
6. The SQL query does the following:

   6.1 Calculates the maximum force towards each parking block using the following formula:

   $F = n(available)/(driving\ time(current\ location, parking\ block) + walking\ time\ (parking\ block, End\ Location))^2$

   Where:
   F = force
   n(available) = number of parking slots available in that block.
   driving time (current location, parking block) = driving time from current user location to the parking block.
   walking time (parking block, End Location) = walking time from parking block to End Location

   6.2 SQL query returns the co-ordinates of the parking block with maximum force and walking time from parking block to End Location.

iii. GREEDY ALGORITHM

Works on the principle of finding the slot with the lease travel time.
Travel Time = Driving Time from current location + Walking time from parking block to End Location

Procedure:

1. Using the Start Location and the End Location get the initial route from the Google Maps API.
2. When the user clicks on “Navigate” button on GUI, get the current location of the user and check if those co-ordinates are intersection points by using the logic described in section 2.1.
3. If the co-ordinates are intersection points, then pass the current user location co-ordinates and Current time to the SQL query.
4. The SQL query does the following:

    4.1 Calculates the travel time as the sum of driving time from current user location and walking time from parking block to the End Location.
    4.2. Returns the parking block co-ordinates with minimum travel time.
    4.3. Query also returns the total walking time from parking block to End Location.

TASK 2: PROBABILISTIC APPROACH

Level of Information Available: Historical Information Available.

Maintenance Cost: Medium Cost

Accuracy: Medium

Probabilistic Database:

The weighted average of availability for three hours, sampled every minute was taken for creating the Probabilistic database. The example below explains this better:

Clock Time	Actual Availability	Probabilistic Availability
0:01	1	1
0:02	4	4
0:03		4
0:04	9	9
0:05		9
0:06		9
Average Availability	**4.66**	**6**

So here in the example where we are considering the availability for first 6 min, for Actual availability: there was parking information for 3 minutes (0:01, 0:02, 0:04). As a result, we get an average of 4.66((9+4+1)/3). So in case of probabilistic, as depicted in the diagram, availability was given to each minute and as a result when we calculate the average now, we get average availability = 6((1+4+4+9+9+9)/6).

The algorithms evaluated using the probabilistic dataset were the same as for deterministic dataset:

i. GRAVITATIONAL FORCE DISTANCE BASED ALGORITHM

ii. GRAVITATIONAL FORCE DISTANCE BASED ALGORITHM

iii. GREEDY ALGORITHM

All three above algorithm have similar implementation specifications as in deterministic data barring the dataset.

TASK 3: BASELINE APPROACH

Level of Information Available: Minimum/ None.

Maintenance Cost: Least Expensive

Accuracy: Low

Procedure:

1. Using the Start Location and the End Location get the initial route from the Google Maps API.
2. When the user clicks on "Navigate" button on GUI, get the current location of the user and check if those co-ordinates are intersection points by using the approximation described in section 2.1.
3. If the co-ordinates are intersection points, then pass the current user location co-ordinates and Current time to the SQL query.
4. The SQL query does the following:

   4.1 Takes as input current user location and parking block ID (initially passed as 0) .
   4.2 Returns the parking block nearest to the current user location.
   4.3 Also returns the parking block ID of the parking block.

5. Then availability of the parking block is checked using deterministic data.
   If the parking block is available to then algorithm terminates, else go to step 3.

EXPERIMENTS

ASSUMPTIONS

Below are the assumptions used for the web based study:

1. The driving speed of the user: 20 miles/hour.
2. The walking speed of the user: 3 miles/hour.
3. The data set that has been considered is from a tourist area in San Francisco, called Fisherman's Wharf. There are 40 nodes and 63 edges in this road network.
4. Each parking location is referred as a parking block. Thus each road segment can have 0, 1 or 2 parking blocks.
5. The midpoint of a parking block is considered to the parking block allotted to the user. Thus user will be navigated to the midpoint of the allotted parking block.
6. Congestion percentage (x%) means elimination of x% of available parking slots at each parking block.

METRICS

In order to evaluate our approach, we designed a web based algorithm study. The aim of the study was to measure the efficiency of above discussed algorithms on a set of user inputs. The comparison was based on following two metrics:

1. **Metric 1:**
   Average (total) time it took for user to reach destination location
2. **Metric 2:**
   Query Time: average time it took an algorithm to find a parking slot.

In total we had 86 valid user inputs for each algorithm, each algorithm had same user inputs including the date and time. Additionally, the date used for these 86 user inputs were spread across the entire week thus covering each day of the week. Similarly, the time used was spread across the entire day. The table shown below shows the day that corresponds to each of these 86 user inputs. The algorithms were also tested by manually varying the congestion level between 0%, 20%, 40% and 60%.

Day of the week	Unique user input set
Monday	13
Tuesday	12
Wednesday	12
Thursday	12
Friday	13
Saturday	12
Sunday	12

FIGURE 1: DISTRIBUTION OF USER INPUTS FOR EACH DAY

APPLICATION

The web based design for user study was developed using following software:

1. Database Server (for storing the application): SQL Server 2012
2. Application Server (for running the web application): Tomcat Server 8.0

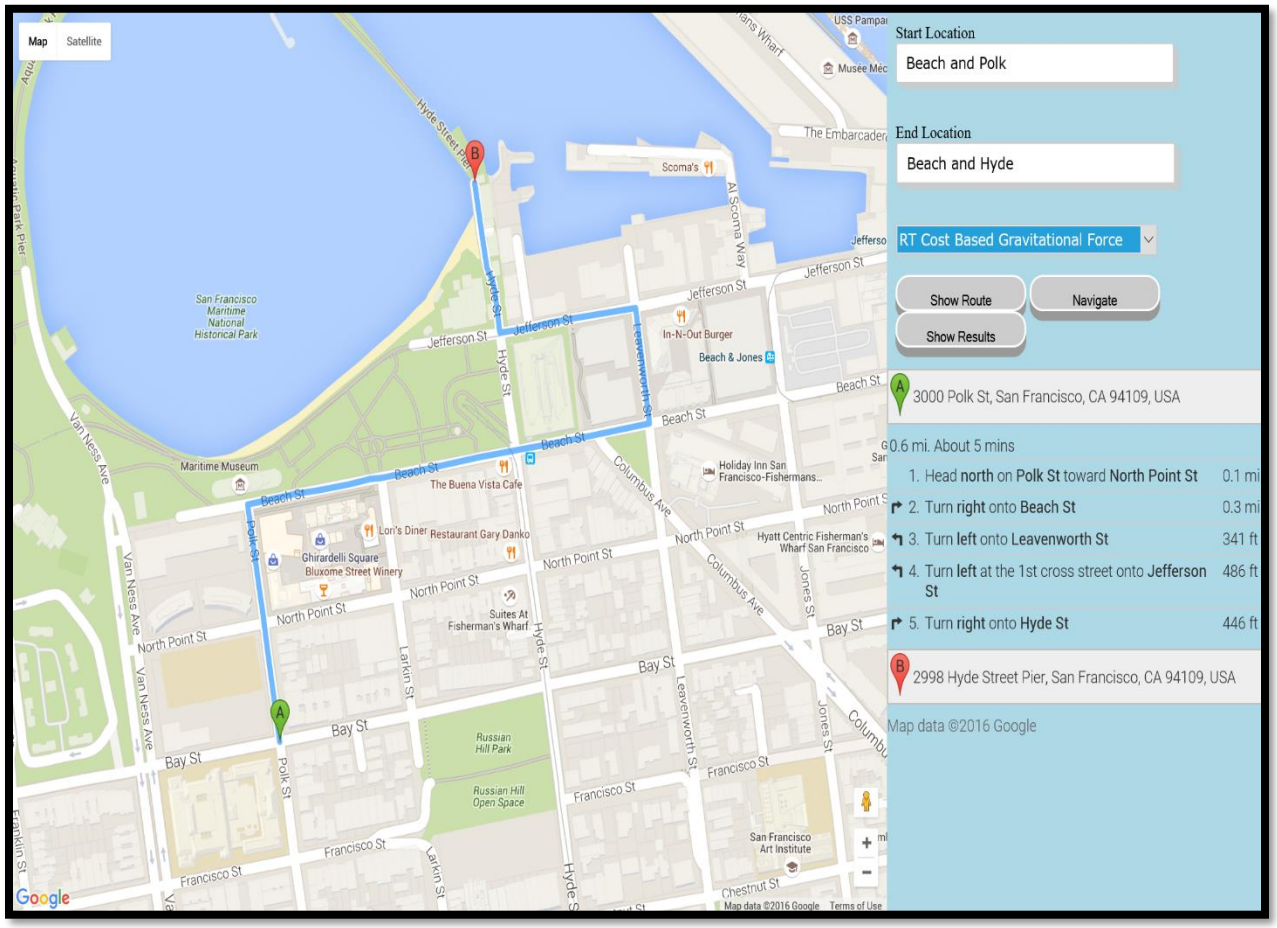


FIGURE 2: A SNAPSHOT FROM THE WEBSITE OF STUDY

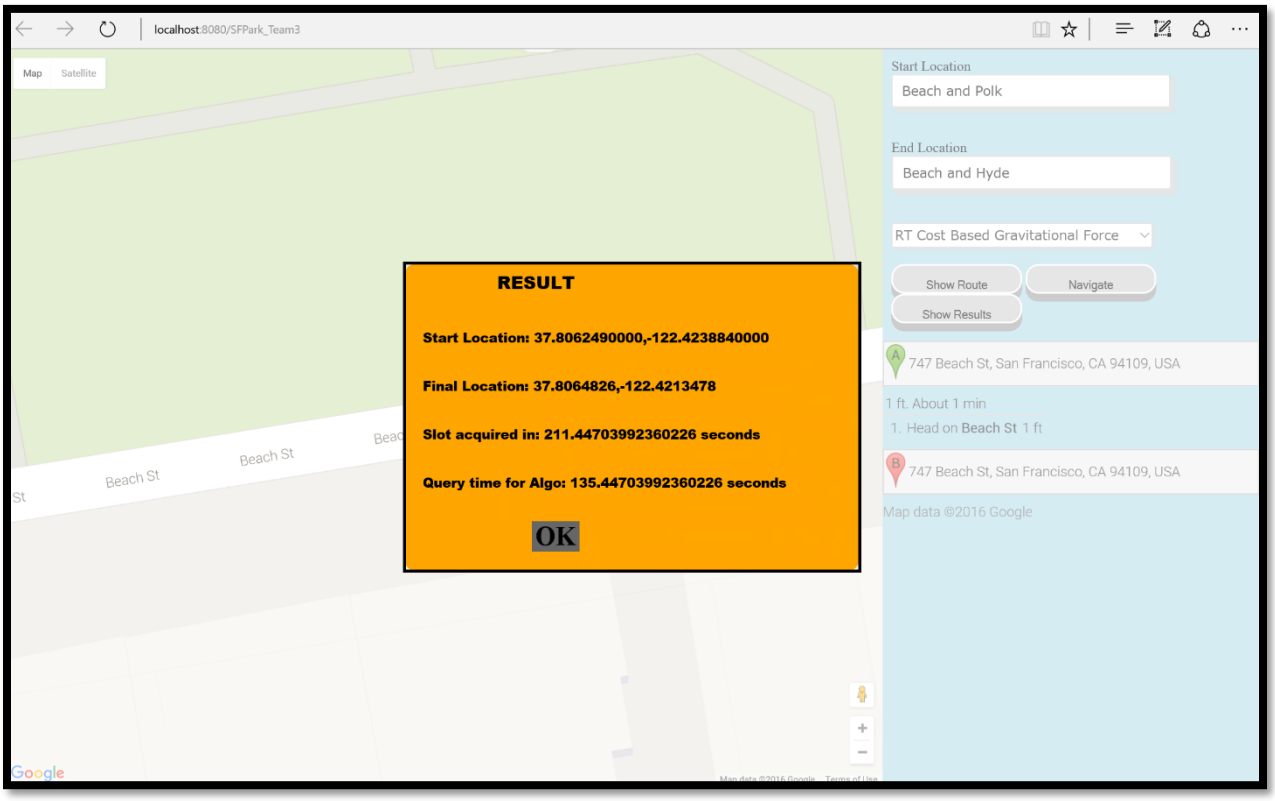


FIGURE 3: A SNAPSHOT FROM THE RESULT WEBSITE OF STUDY

Figure 2 shows a snapshot from the website of study. As we can see in Figure 2 and Figure 3, the inputs provided for each application are:

1. User's current location/Start Location.
2. User's destination.
3. Current Date & Time of Search.
4. Algorithm to be evaluated.
5. Click-able buttons by which user can start the navigation from user's current location to the allotted parking block by the algorithm.
6. Show Results button, which enables user to view the above two metrics of the algorithm.

RESULTS

As discussed in earlier sections, the evaluation was to be done for the different task where in the data set limitations varied. These tasks were categorized as Deterministic, Probabilistic and Baseline approach. Our study was done to compare efficiency of each algorithm that was considered for each of these studies first and then combine the best of each study to compare efficiency of best of each category. The evaluation results are presented below in the same sequence along with the observations and justification of these results.

DETERMINISTIC APPROACH

METRIC 1:

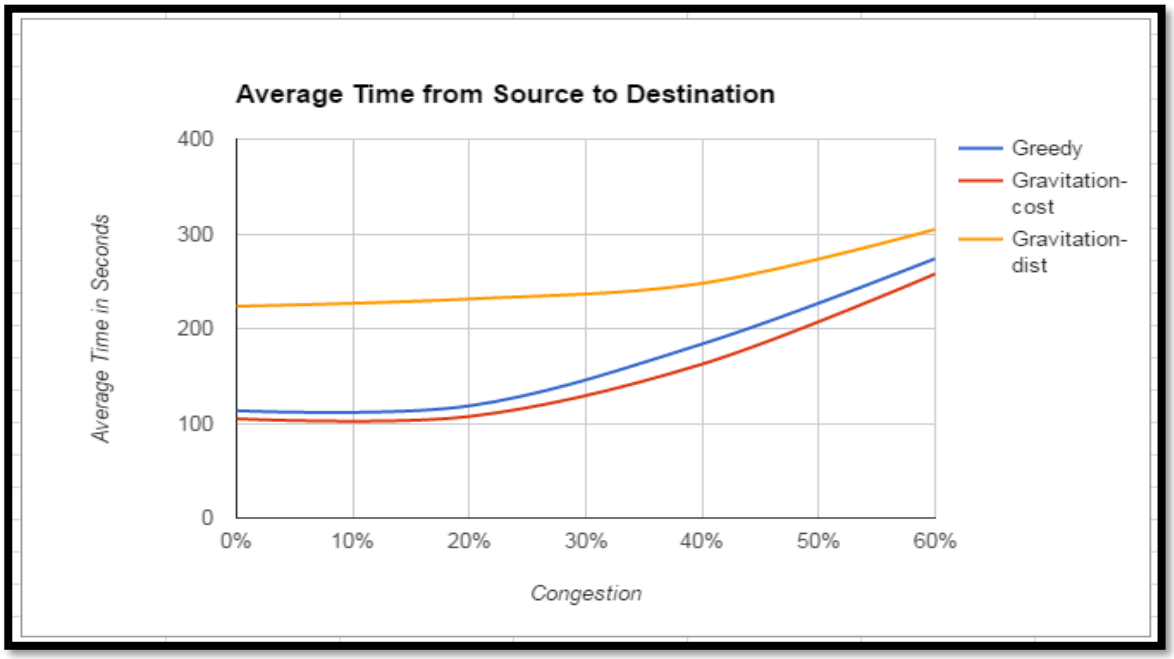


FIGURE 3: DETERMINISTIC APPROACH: METRIC 1

The figure above shows the evaluation of three deterministic algorithms for metrics 1. The x axis of the graph plot shows the variation in the congestion level and the y axis shows the Average time it took the user to reach user's destination from start location (in sec). Important Note- the Average time means driving time to the allotted parking block plus the walking time from the parking block to user's destination location.

Observation and Justification:

1. So we can see, distance based gravitational algorithm has shown a poor performance compared to the other two algorithms. This is because the distance based just takes into the account the maximum force it gets as an output allotting user with a slot which can be even far away from the user's destination. Thus the overall performance degrades since in many cases the when

walking time is added to time required to fetch the available slot, the total time increases for distance based algorithm.

2. Cost based Gravitational force has shown the best performance among all three algorithms which is closely followed by greedy algorithm.
3. Cost based Gravitational algorithm has an edge over greedy algorithm because it makes sure that user is allotted a parking block which has highest possibility of getting a parking and hence reduces the possibility of a miss or re-routing when the user is about to reach the allotted parking block.
4. Now greedy algorithm only accounts the total travel time and not the possibility of the miss or re-route when the user is about to reach the allotted parking block. Thus, at times greedy algorithm leads to re-routing in case the parking block allotted gets filled up.
5. For cost based gravitational algorithm as well as greedy algorithm, the average time to reach user's destination is near to 125 seconds between 0-30 percent of congestion and then for congestion percent above 30 both the algorithms start to degrade in performance as the available parking slots starts decreasing. As shown in graph, the average time to reach user's destination increases to almost 240 seconds for 60% congestion.

METRIC 2:

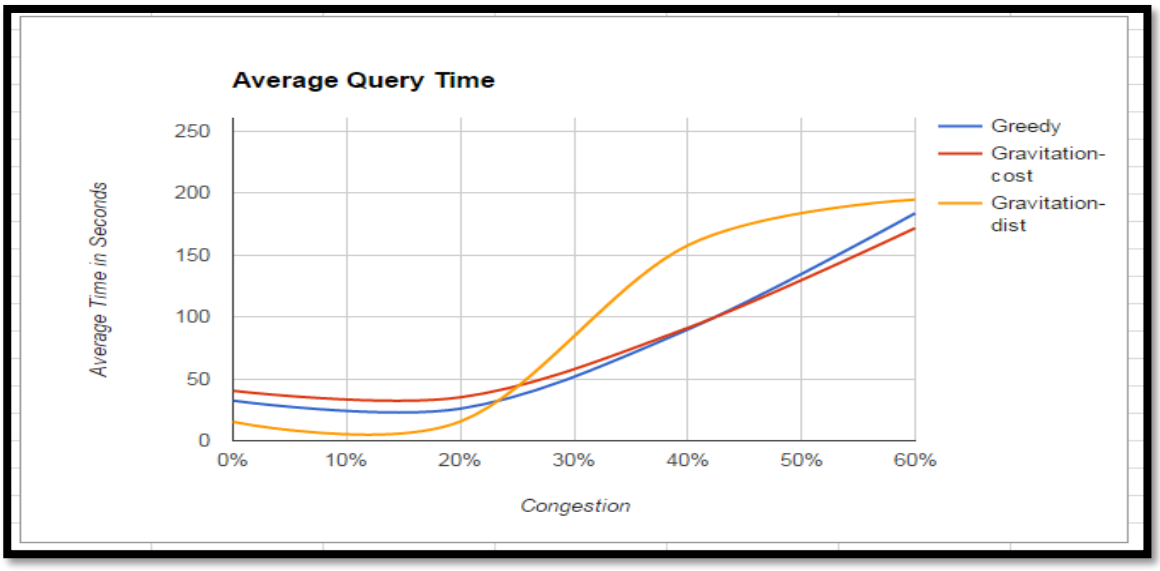


FIGURE 4: DETERMINISTIC APPROACH: METRIC 2

The above figures show the evaluation of three deterministic algorithms for metrics 2. The x axis of the graph plot shows the variation in the congestion level and the y axis shows the average query time (i.e. the time required to fetch the available slot)

Observations and Justification:

1. Here, distance based gravitational algorithm shows a good performance compared to other two algorithms with low congestion percentage. This is because with distance based, the query can fetch immediately the nearest blocks which has the maximum number of slots available and it does not consider the user's destination.
2. From Figure 4 we can see that initially the distance based performs better than both cost based gravitational and greed, but as the congestion level increases, the performance of distance based decreases whereas greedy and cost based performance are nearly same.

3. Thus, overall Cost based and Greedy performs well even with high congestion level (60%) with the average query time as still 165 seconds.

PROBABILISTIC APPROACH

METRIC 1:

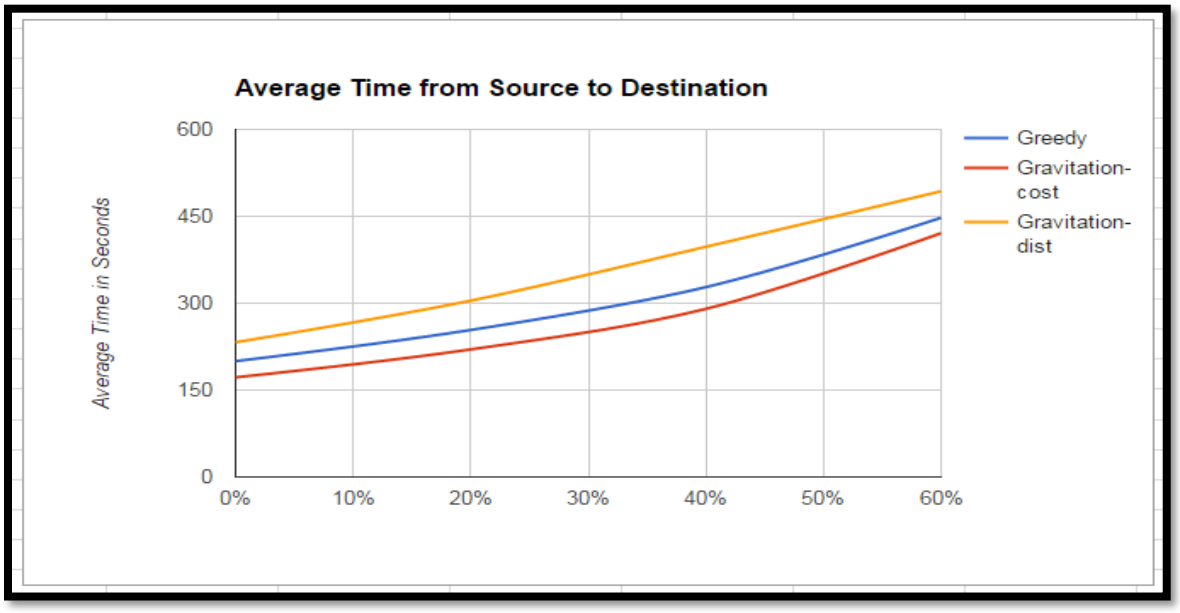


FIGURE 5: PROBABILISTIC APPROACH: METRIC 1

The Figure 5 above shows the evaluation of three Probabilistic algorithms for metrics 1. The x axis of the graph plot shows the variation in the congestion level and the y axis shows the Average time it took the user to reach user's destination from start location (in sec). Important Note- the Average time means driving time to the allotted parking block plus the walking time from the parking block to user's destination location.

Observations and Justification:

1. The results above for deterministic approach shows: the distance based algorithm has a poor performance compared to other algorithms since it does not account user's destination. And as the congestion level increases, the performance degrades badly.
2. For deterministic approach, the cost based gravitational and greedy shows the same kind of performance but as the congestion increases, the performance degrades as the availability is decreased and the algorithms work on probabilistic data rather than real time data.
3. Greedy algorithm doesn't really perform as it performed for deterministic approach as with the probabilistic data the chances miss/re-route increases.
4. The average time to reach user's destination increases almost as high as 450 seconds for all these algorithms with congestion level of 60%.

METRIC 2:

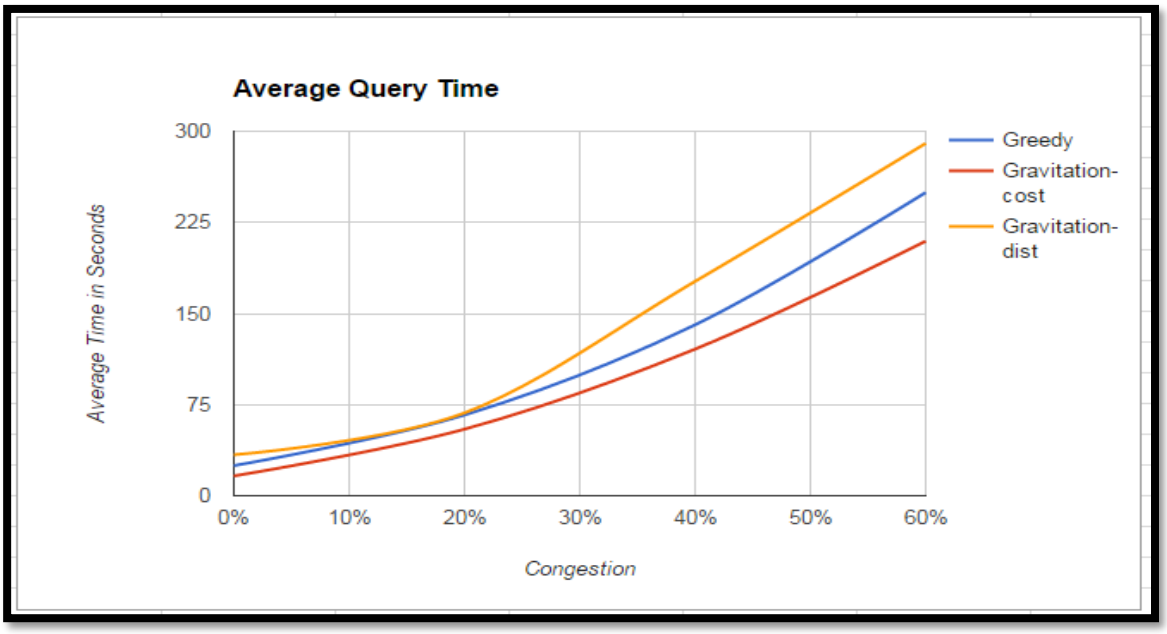


FIGURE 7: PROBABILISTIC APPROACH: METRIC 2

The above Figure 6 shows the evaluation of three deterministic algorithms for metrics 2. The x axis of the graph plot shows the variation in the congestion level and the y axis shows the average query time in seconds.

Observations and Justification:

1. As discussed for previous category, again distance based gravitational shows poor performance compared to other two algorithms in this category also with an average query time near to 30 seconds at 0% congestion level.
2. Again cost based gravitational has shown a better performance than greedy, this is because the number of re-routes that comes with cost based are very less then when compared to the re-routes by greedy algorithm.

3. As the congestion level increases, the worst performance is shown by Distance based and the average query time is nearly 300 seconds which is nearly 5 minutes for just querying and fetching the results. This overall degrades the performance of distance based.

4. The Cost based and greedy have a good average query running time even with congestion level at 30%. But as the congestion increases, the average query time is more than 200 seconds, nearly 3 minutes, which is very slow.

BASELINE ALONG WITH OVERALL PERFORMANCE

METRIC 1:

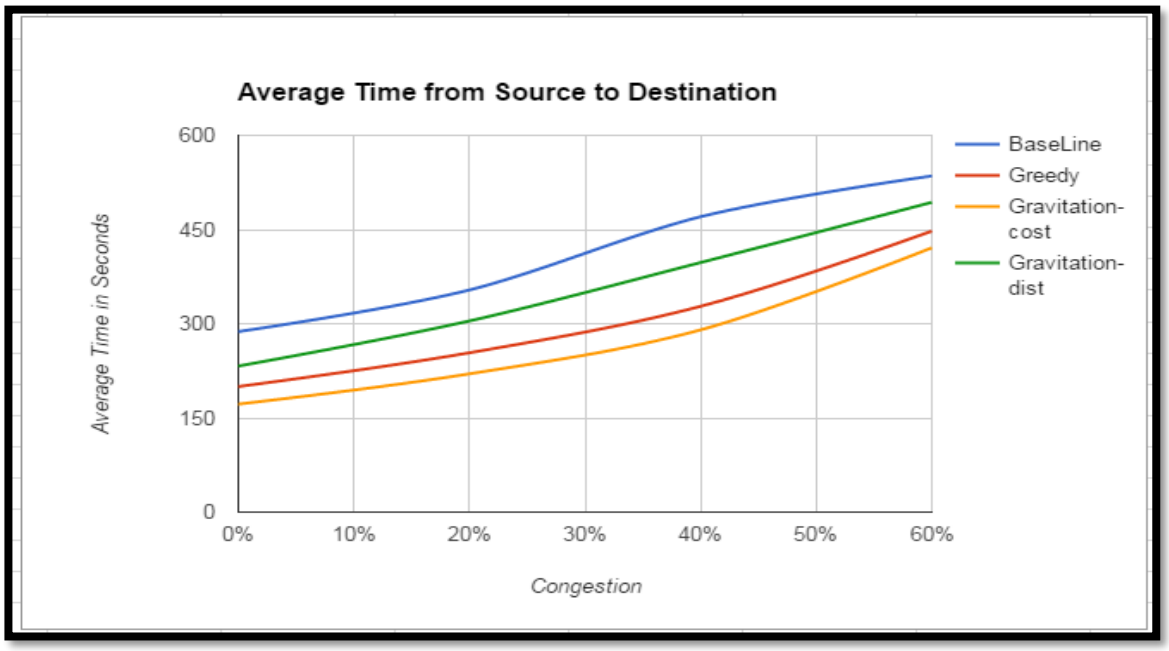


FIGURE 7: BASELINE: METRIC 1

The Figure 7 above shows the evaluation of the best algorithms evaluated for three categories for metrics 1. The algorithms that are considered for overall performance comparisons are: Greedy algorithms (Deterministic approach), Distance based gravitational (Deterministic approach), Cost based gravitational algorithm (Probabilistic approach) and Greedy algorithms (Baseline approach). The x axis of the graph plot shows the variation in the congestion level and the y axis shows the Average time it took the user to reach user's destination from start location (in sec). Note-the Average time means driving time to the allotted parking block plus the walking time from the parking block to user's destination location

Observations and Justification:

1. As expected, the base line greedy algorithm shows the worse performance of all the algorithms as this approach has very narrow data set access and hence shows poor performance. The average time is around 300 seconds for congestion level 0% which is nearly 5 minutes.
2. Deterministic approach has an edge over Probabilistic approach. This is because the number of re-routes for Deterministic are minimal.
3. The Distance bases still performs better than Baseline Greedy approach even for high congestion level.
4. The performance degrades as the congestion increases but still the performance of Greedy and Cost based for deterministic and probabilistic approach is still good at high congestion level. Thus both have an edge over distance based and baseline approach.

METRIC 2:

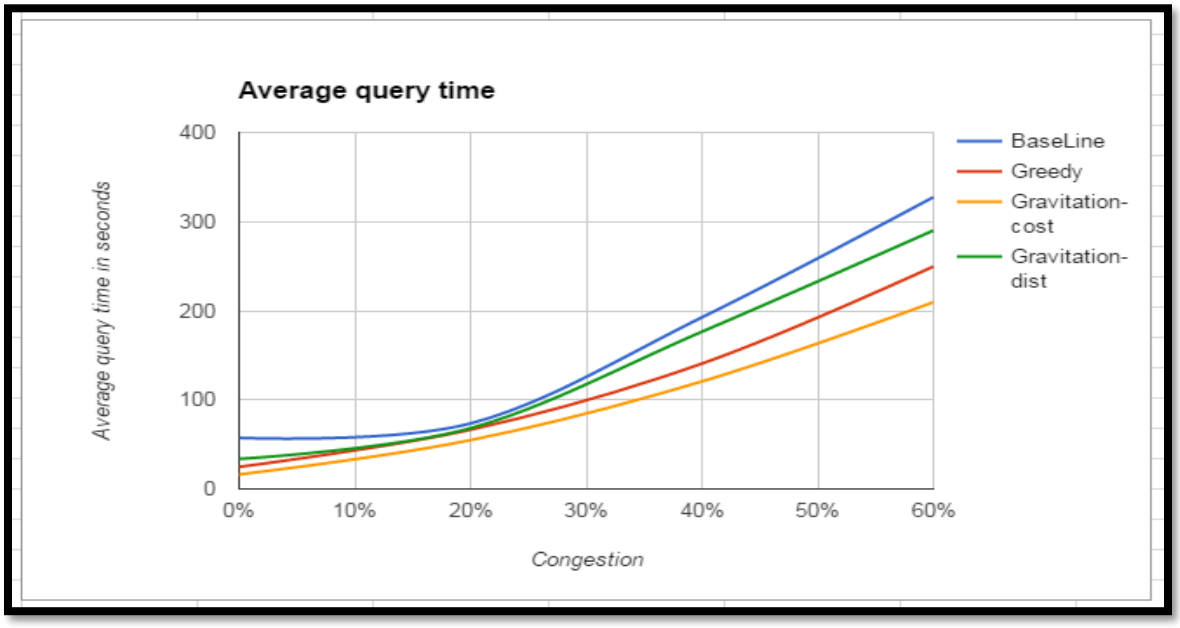


FIGURE 8: BASELINE: METRIC 2

The Figure 8 above shows the evaluation of the best algorithms evaluated for three categories for metrics 2. The algorithms that are considered for overall performance comparisons are: Greedy algorithms (Deterministic approach), Distance based gravitational algorithm (Deterministic Approach), Cost based gravitational algorithm (Probabilistic approach) and Greedy algorithms (Baseline approach). The x axis of the graph plot shows the variation in the congestion level and the y axis shows the average query time for giving the response i.e. the empty parking slot.

Observations and Justification:

1. As seen from the graph, the Baseline algorithm still has the worst performance.
2. Initially for a low congestion level, the three algorithms have nearly the same performance whereas Baseline still performs badly with an average query running time of 60 seconds.
3. Comparing the probabilistic approach and real time approach, the probabilistic approach still performs better with the extrapolated data showing that even with the re-route, the query to fetch the nearest empty slot is still high.

FUNCTIONAL DESIGN

DESIGN

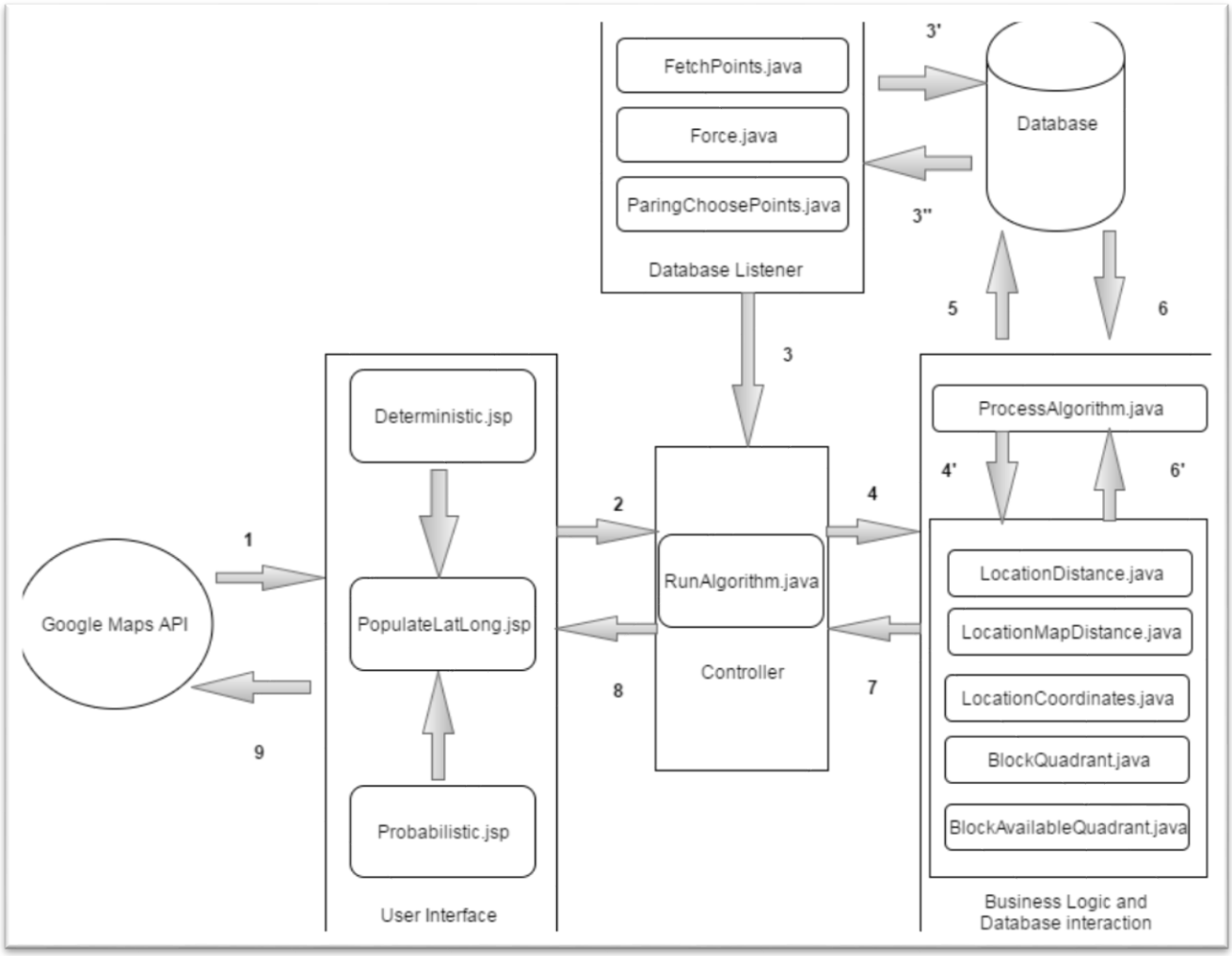


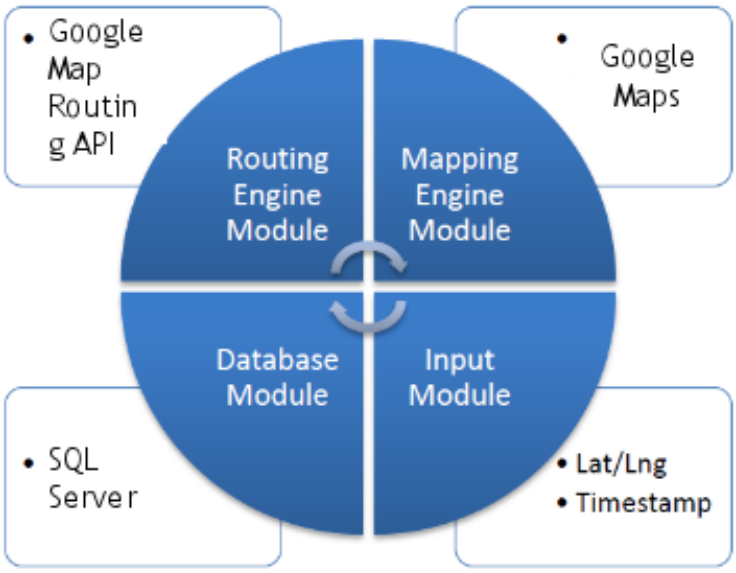
FIGURE 10: APPLICATION DESIGN

Key:

1. Point A: Point A will be initial start point
2. Point C: Point C will be the current position of the vehicle (But in the actual application this will be represented by marker A)
3. Point B: Point B will be the intended destination
4. Point P: Point P will be the parking slot
5. Point I: Point I will be the road intersection point

Modules:

The application consists of below major modules:



1. Google Maps API:

Google Maps API is used for the following

- Getting the initial route between A and B
- Updating the route between C and allocated P
- Moving the vehicle along the route from C to allocated P
- Getting all the distances between all I and all P
- Getting driving time between all I and all P
- Getting walking time between all I and all P

2. User Interface:

Display module is used to implement the UI of the application and consists of three JSPs

- Deterministic.jsp: Displaying the UI for real time algorithm implementation
- Probabilistic.jsp: Displaying the UI for probabilistic and baseline algorithm implementation
- PopulateLatLong.jsp: This helps in fetching all the node names from the database. As the user keys in one or more alphabets, any node name starting with that is auto populated in the drop down allowing the user to select the start and end destination
- Display module takes the inputs (A or C, B and algorithm name) from the user and passes it on to the Controller module
- Display module also takes destination parking slot co-ordinate returned by the controller module and then passes these co-ordinates to the Google Maps API to update the route between A and destination parking slot location

3. Controller:

- Controller consists of a servlet RunAlgorithm.java
- Controller basically acts as interface between the Display module and the Implementation module
- This module takes as input A or C, B and algorithm name
- Controller takes as input every point on the route as position of C keeps changing and checks whether that point is an intersection or not.
- If that point is an intersection, then Implementation module is called else user moves ahead to the next point on the route
- Checking whether the points is an intersection or not is done using the Database Listener Module.
- Also the approximation of intended user destination B to the nearest parking block midpoint is done using Listener Module

4. Implementation:

- This is the most important module and this is the module where algorithm executes
- This module consists of
    a. ProcessAlgorithm.java: Java class that executes all the algorithms
    b. DB Helper Module: Consists of various helper classes that help ImplementAlgorithm.java in implementing the algorithm
    c. This module takes as input A or C, B and algorithm name and then executes the algorithm by querying the Database and using DB Helper classes
- This module returns the co-ordinates of the allocated P to Controller.

5. Database Listener:

- This module is invoked only once when the Tomcat server is started
- The main function of this module is to check the points where we need to make a call to the ProcessAlgorithm Module (this is done to trigger algorithm at all the decision making/intersection points) and to approximate D to the closest midpoint of the parking block
- This module has the following classes
    a. FetchPoints: extract all nodes in the database
    b. ParkingChoosePoints: these points are basically the mid-points of all the parking blocks
    c. Force: This class performs two main functions:
        - Assigns weight to all the nodes depending on their distance from the point C. The node with maximum weight is thus the point where Implement Algorithm needs to be called.
        - The above mentioned logic is similarly applied to approximate D as midpoint of the parking block
- So, this module populates all the intersection points and parking block midpoints in one go i.e. during server startup.
- This greatly reduces the number of calls we make to the database and thus increasing the efficiency.

FLOW

On Deterministic.jsp user enters A and B and clicks on “Show Route” button.

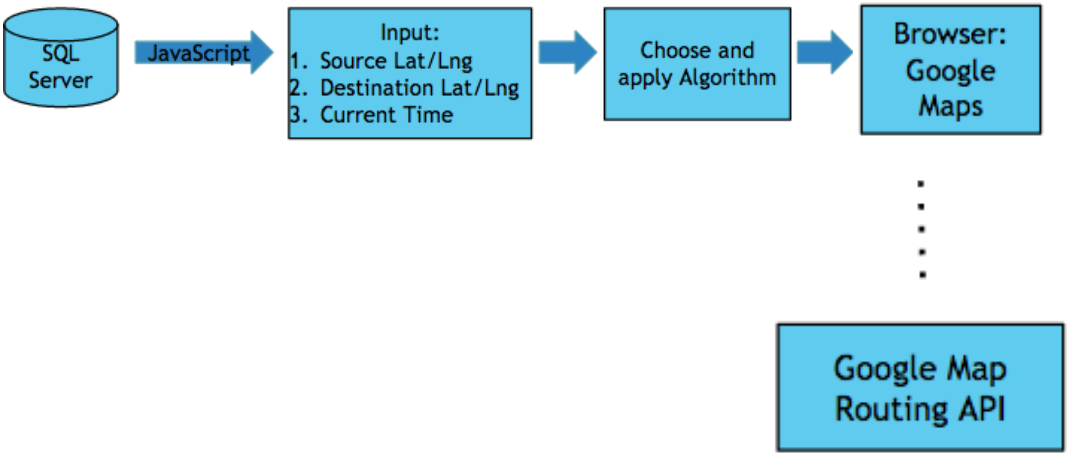
1. A and B co-ordinates are passed to Google Maps API and the API returns an initial route to the user

   As the user clicks on navigate button marker a marker at initial start position A starts moving towards B along the route

   Following sequence of steps are executed as user moves to each point on the route presented by the Google Maps API

2. The Display Module passes A or C, B and algorithm name to the Controller Module
3. Controller Module checks whether passed A or C is an intersection points. If passed A or C is an intersection point, then next step 4 is executed, else the user moves to the next point on the route presented by Google Maps API.
4. Implementation Model is invoked and algorithm is executed based on the algorithm name passed. DB Helper classes are used to implement the algorithm.
5. Implementation Model makes a call to the database by passing A or C and B
6. Database returns an updated P (Location of Parking Slot) to the Implementation Module.
7. Implementation Module forwards the received P to the Controller Module
8. Controller Module Sets B = P (Intended destination as the location of the parking slot) and sends B to Display Module

Display Module sends B to Google Maps API, and Google Maps API returns an updated route to the user based on the parking slot returned from the Database



SIMULATION

SOFTWARE NEEDED

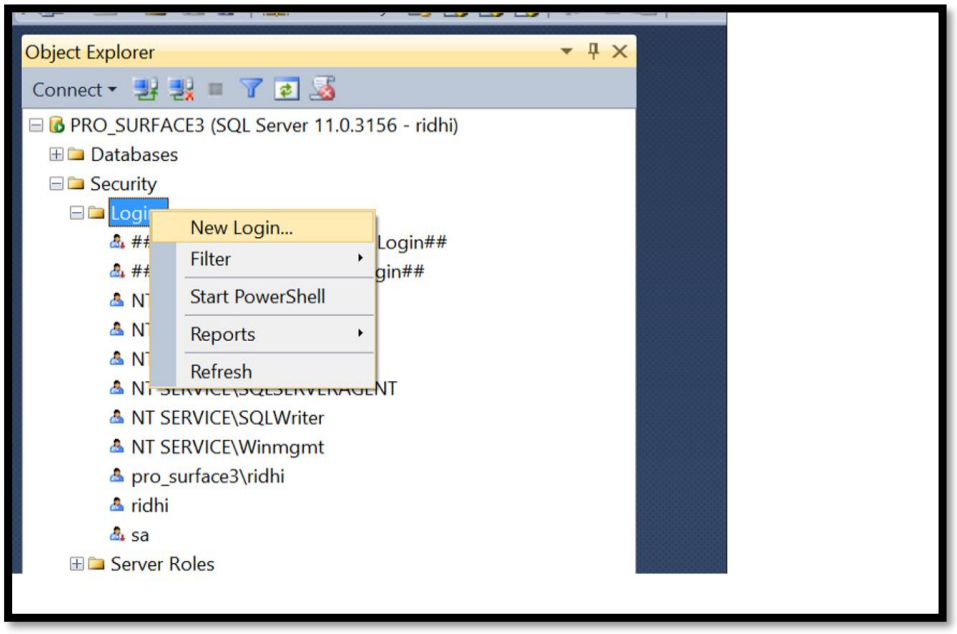
Following software are needed for running the application:

1. Database Server (for storing the application): SQL Server 2012.
2. Application Server (for running the web application): Tomcat Server 8.0

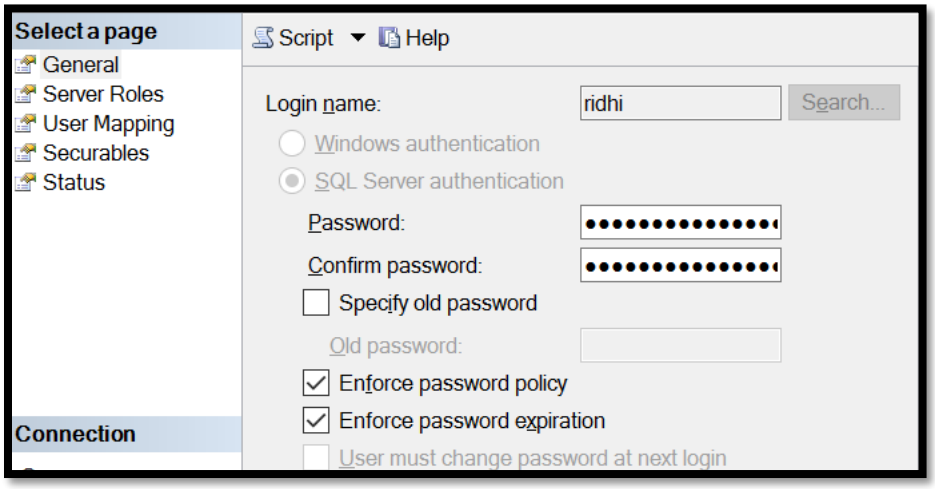
CONFIGURING SQL SERVER 2012 WITH TOMCAT 8.0

Steps for configuring SQL Server 2012 with Tomcat 8.0

a. Install Microsoft SQL Server 2012 with default configurations.
b. Copy the database file (Team3_SFPark.bak) of the project your account folder which is located in the C:\Program Files\Microsoft SQL Server\MSSQL11.MSSQLSERVER\MSSQL\Backup\
c. Connect to the database in the SQL server using web authentication mode.
d. Expand the project and right click the Databases and select restore Database option.
e. In the General tab, select Device option and then select the "…" option on right side.
f. A dialog box appears, click on Add button.
g. Traverse to the path mentioned above and select "Team3_SFPark.bak"
h. Click OK for all subsequent dialog boxes
i. Disconnect the database and reconnect using web authentication or restart the database connect.
j. Expand the project and then "Security" and right click on "Logins" and select "New Login…"



k. Select “SQL Server authentication” radio button and write name: “ridhi” and password: “ridhi129” and check all boxes as depicted in below diagram and then click on “OK”



l. Refresh security and then right click on the user “ridhi” and give the admin rights to this user.
m. Right Click on the database and set authentication mode to both windows as well as SQL Server.
n. Restart the database engine.
o. Database is now setup.

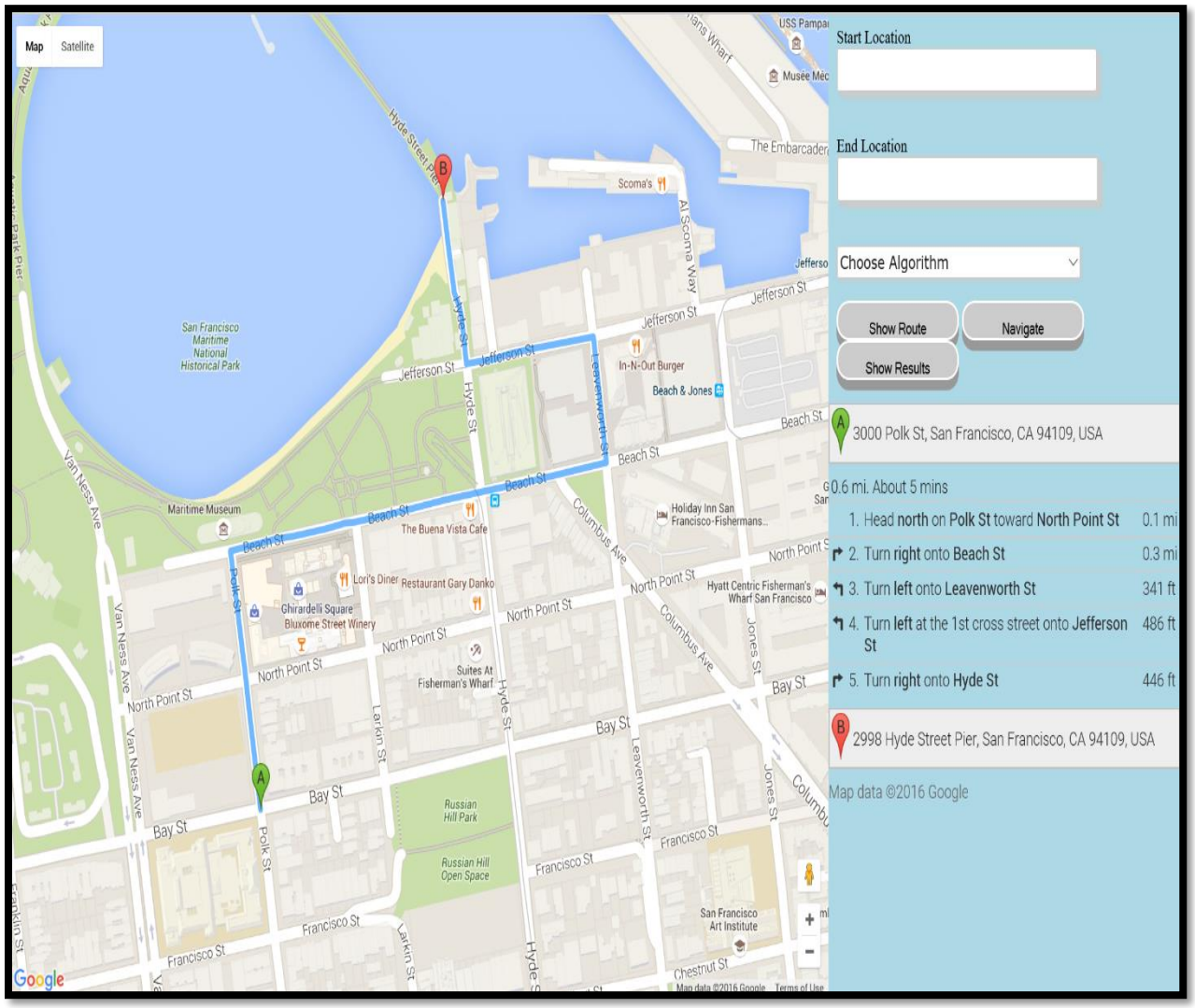
RUNNING SIMULATIONS

Steps for running the application

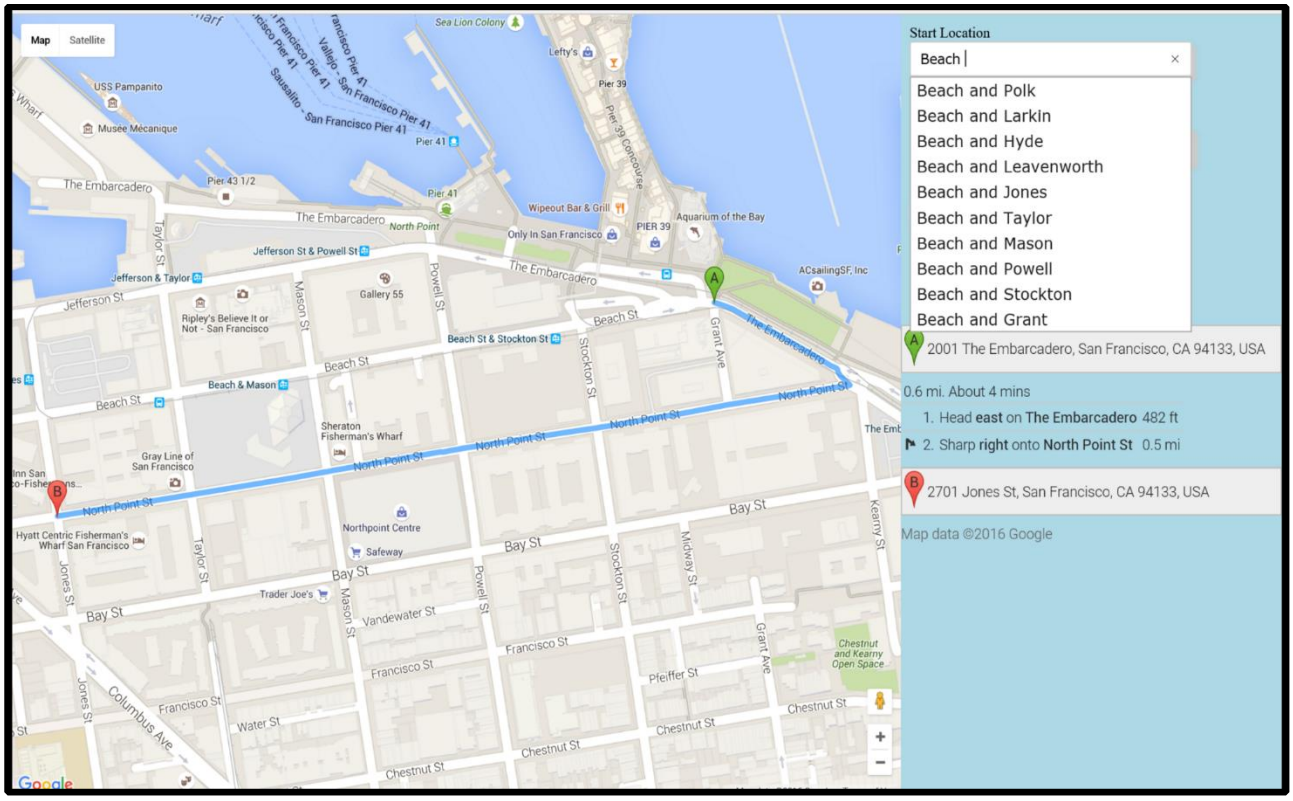
1. Download tomcat 8 server from the following link https://tomcat.apache.org/download- 80.cgi
2. Copy and paste the zip file
3. Unzip the zip file at some location. Let us call your folder “<tomcat8>”
4. Under your <tomcat8>/webapps copy the CS581_SFPark_Team3_2016.war
5. If server is running stop the server by running the shutdown.bat file under <tomcat8>/bin folder
6. Then start the server by running startup.bat Under <tomcat8>/bin folder
7. Access the application using http://localhost:8080/CS581_SFPark_Team3_2016
8. For accessing the deterministic algorithm use the link http://localhost:8080/ CS581_SFPark_Team3_2016/
9. For accessing the probabilistic and baseline algorithm use the link http://localhost:8080/ CS581_SFPark_Team3_2016/Probabilistic.jsp

SAMPLE RUN

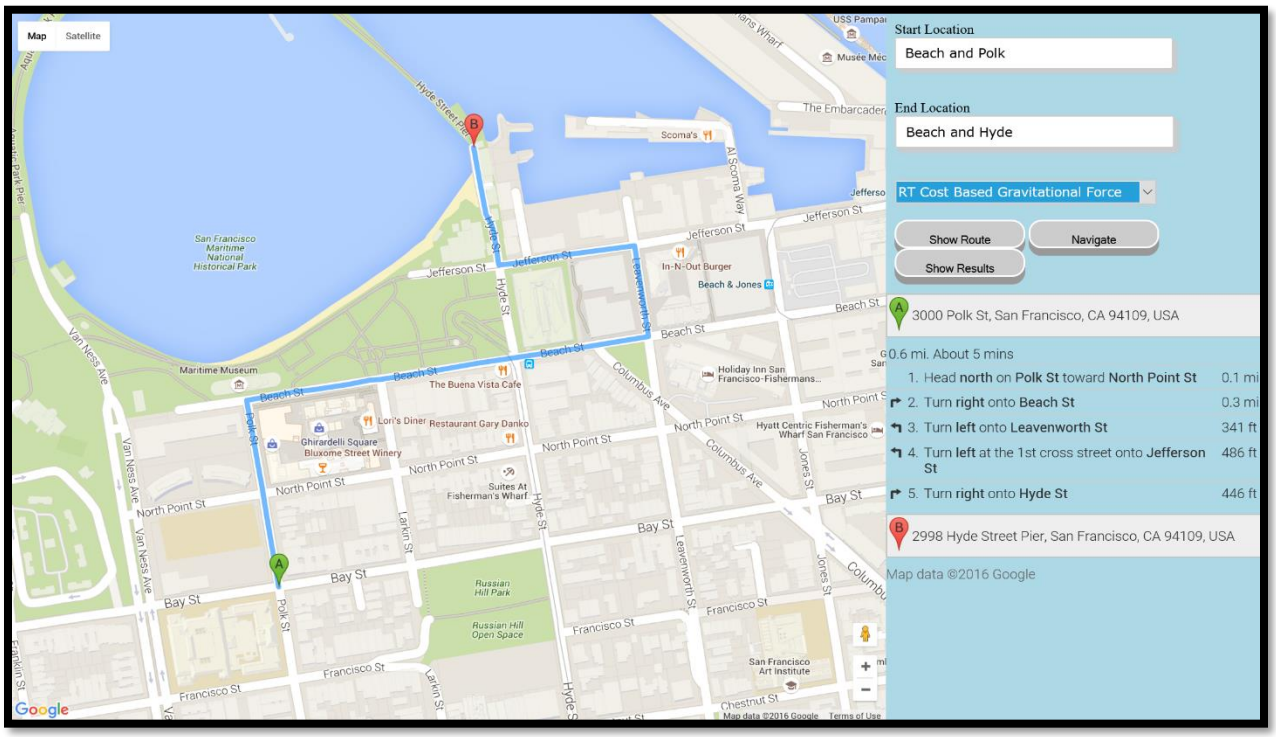
1. Use http://localhost:8080/CS581/SFPark_Team3_2016 to launch the application on a web browser:

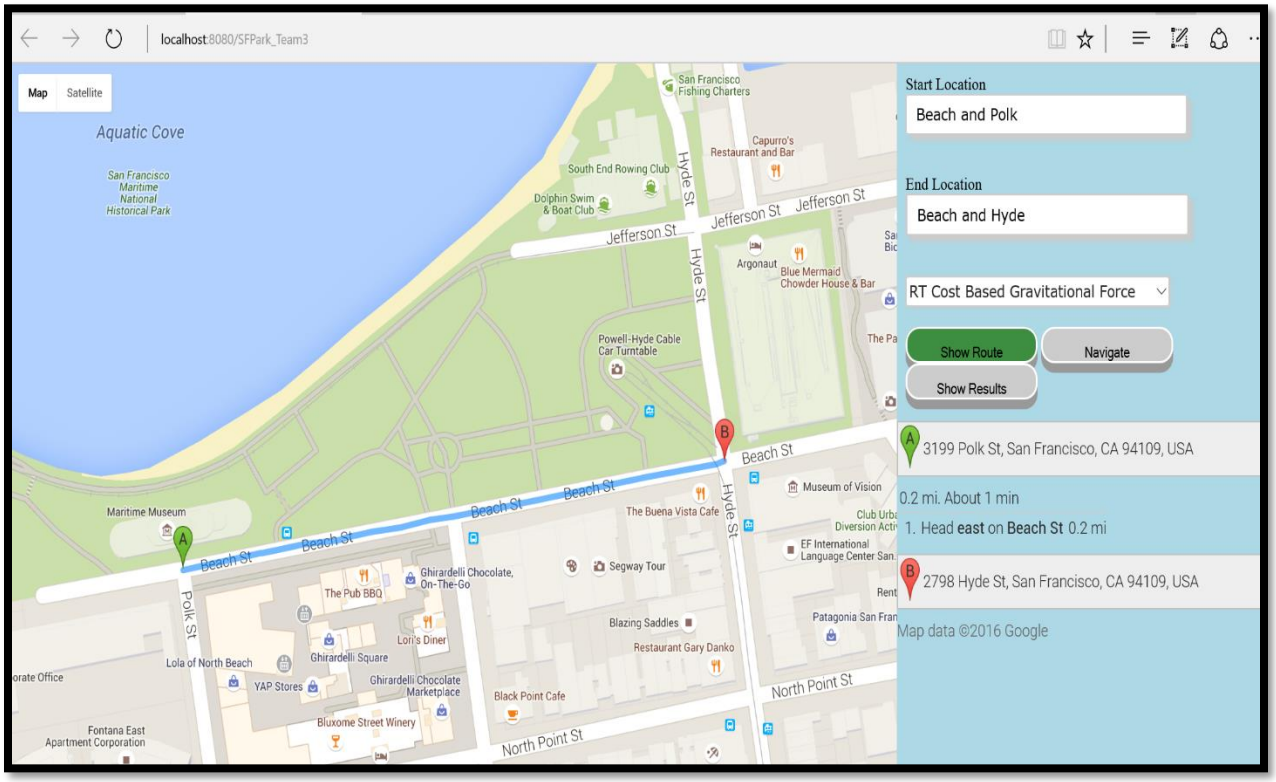


2. Select the start and end locations from the drop down which appears when you enter the first letter of the location name.

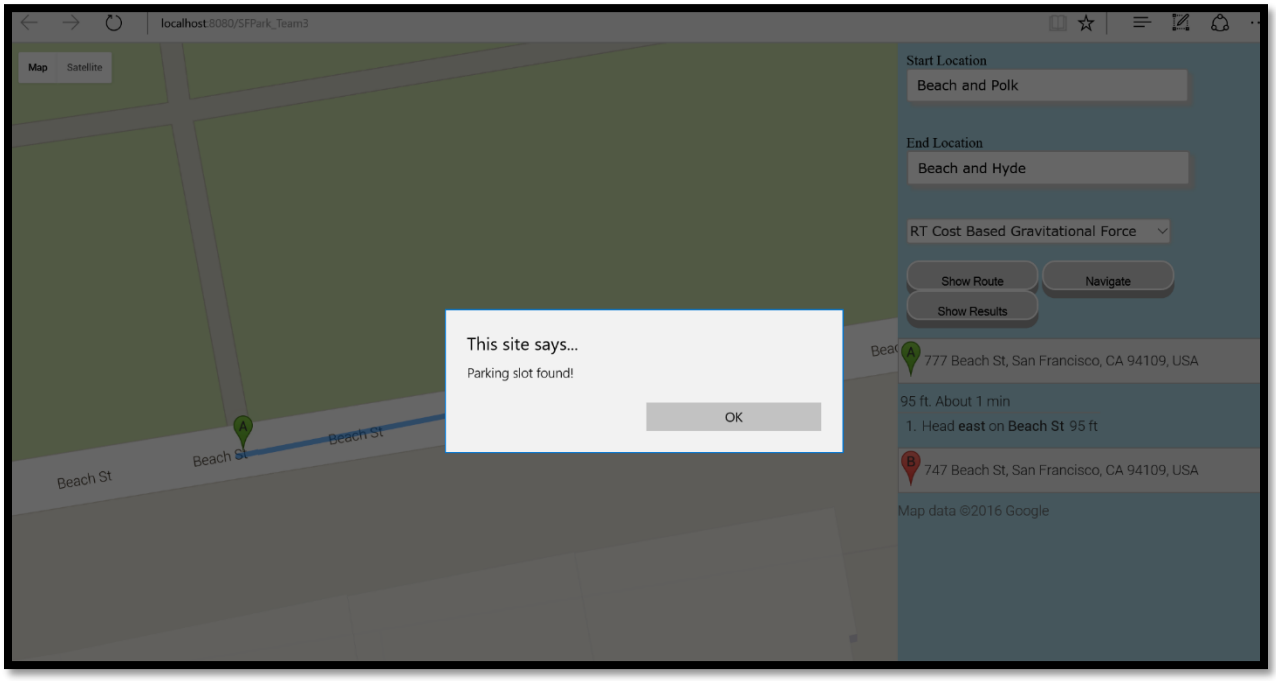


3. Select desired algorithm and then click on “Show Route” followed by “Navigate” to start the execution of the algorithm.





4. Once the parking slot is acquired, the user is notified by a popup box.



5. Once the user reaches the destination, then chick on show "Show Results" to get the user final location, time elapsed and the time taken by the chosen algorithm to find the acquired parking slot.