

# Search of Spatio-temporal Resources: Final Project Report

Group# 2

## **Team Members:**

Jay Dave

Prateek Garg

Kartikey Pradhan

Kunal Thorvat

# Contents

Abstract.....	3
Algorithms Evaluated.....	4
Adaptation to Road Network .....	4
Algorithms.....	4
Experiments .....	10
Results.....	12
Deterministic Approach .....	12
Probabilistic Approach .....	15
Baseline along with overall performance .....	17
Functional Design .....	19
Design .....	19
Flow.....	22
Simulation .....	23
Software needed .....	23
Configuring SQL Server 2012 with Tomcat 8.0.....	23
Running Simulations .....	24
Sample Run.....	24

# 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.

We simulated the experiments on a web-based interface, running on Tomcat 8 server and SQL Server 2012 used as the database server. The simulations assume the given time-stamped availability data as the real-time data, with no information to the future data.

So we had to perform the following tasks

1. Task 1: 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: 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: To find a parking spot for a user in an uninformed search with only information of user destination, starting location and parking block.

The results are presented from 86 simulations in each implementation of each task, and each congestion level. The graphs are also presented depicting the performance of the algorithms in each of the mentioned scenarios.

The results are reproducible with the code provided (instructions for rerunning the simulations are included).

The results of the experiments indicate that having the access to real time parking availability information increases the efficiency of the algorithms in all variations. The less information we have, the worse the results are (time taken to find a parking spot increases relatively).

Also, the results indicate for lesser congestions, the greedy algorithm performs comparable to the other algorithms, but as the congestion levels increase, the efficiency falls. This indicates that using more sophisticated algorithms like Gravitational Force algorithms is a better design for overall robustness of the system

# Algorithms Evaluated

## Adaptation to Road Network

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

Following is the behavior of all our algorithms as the user moves along the road network:

All our algorithms will be invoked as soon as 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. But, the road intersection co-ordinates provided by the database did not match the road intersection co-ordinates provided by the database. So we had to perform a few approximations to identify the road intersection points and then trigger our algorithms.

We did the following approximations to estimate the road intersection points

- Initially we take all the intersection nodes from the database so that for intersection approximation, we have less database interaction.
- Similarly we do the same for the midpoint of the parking blocks. Midpoint of the parking block is out approximated point where parking slot is located on the block.
- We take in the starting co-ordinates of the user and approximate it to nearest intersection node by using the distance formula:

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

- A point with minimum distance is given max weight and this point is made the approximated user starting location. This is done as our algorithm is triggered at each intersection node.
- 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.

## Algorithms

Following is the list of the algorithms that we have evaluated based on different tasks:

### Task 1: Deterministic Approach

## GRAVITATIONAL FORCE DISTANCE BASED ALGORITHM

Algorithm works in the following manner:

### **Inputs:**

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

### **Steps:**

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:
  - a. Calculates the maximum force towards the each parking block using the following formula

$$F = n \text{ (available)} / (\text{distance}(\text{current location, parking block}))^2$$

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

- b. 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.

### **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)

## GRAVITATIONAL FORCE COST BASED ALGORITHM

Initially we implemented the algorithm which calculated the Gravitational force on the basis of availability at parking block and the Cost → (Driving time + Walking

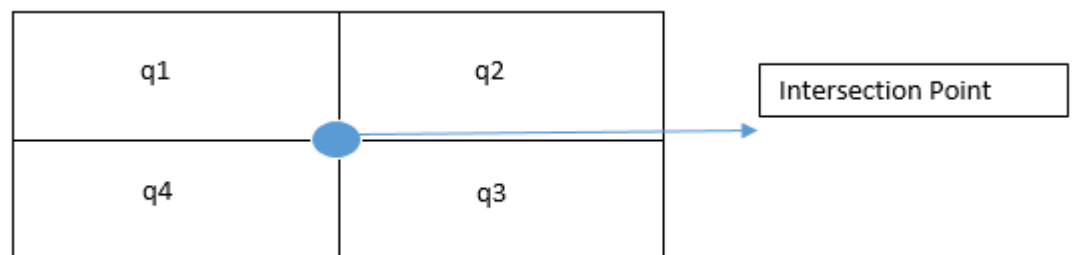
time). But later we tweaked our approach and observed that simulation results improved. We implemented the aggregated gravitational force calculation formula for finding parking slot to user. In the new implementation, we trigger the algorithm at each node. At each node, we are divided the parking block availability with respect to four quadrants as in case of a 2-D graph. Algorithm works in the following manner:

**Inputs:**

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

**Steps:**

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 use the current user location and Current Time to form 4 quadrants as shown below



4. Get the quadrant with maximum availability
5. Pass the co-ordinates of the parking block with maximum availability to the SQL query
6. The SQL query does the following:
  - a. Calculates the maximum force towards the each parking block using the following formula
$$F = n(\text{available}) / (\text{driving time}(\text{current location, parking block}) + \text{walking time}(\text{parking block, End Location}))^2$$

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

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

**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)

## GREEDY ALGORITHM

Algorithm works in the following manner:

**Inputs:**

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

**Steps:**

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

**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)

## Task 2: Probabilistic Approach

### Probabilistic Database:

Probabilistic database was created using the weighted average of availability for three hours, sampled every minute. To elaborate more on this point, here is an example:

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
<b>Average Availability</b>	<b>4.66</b>	<b>6</b>

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 =  $6((1+4+4+9+9+9)/6)$ .

Following algorithm were evaluated for the probabilistic approach

### GRAVITATIONAL FORCE DISTANCE BASED ALGORITHM

Same as described for deterministic approach

### GRAVITATIONAL FORCE DISTANCE BASED ALGORITHM

Same as described for deterministic approach

### GREEDY ALGORITHM

Same as described for deterministic approach

## Task 3: Baseline Approach

Following algorithm was evaluated for baseline approach:



## BASELINE ALGORITHM

Following is the working of the algorithm:

### **Inputs:**

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

### **Steps:**

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:
  - a. Takes as input current user location and parking block ID (initially passed as 0)
  - b. Returns the parking block nearest to the current user location
  - c. Also returns the parking block ID of the parking block
5. Then availability of the parking block is checked using deterministic data.
6. If the parking block is available to then algorithm terminates, else go to step 3

### **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)

# Experiments

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 :**  
Average Distance between the parking slot allotted and user destination or average walking time for the user

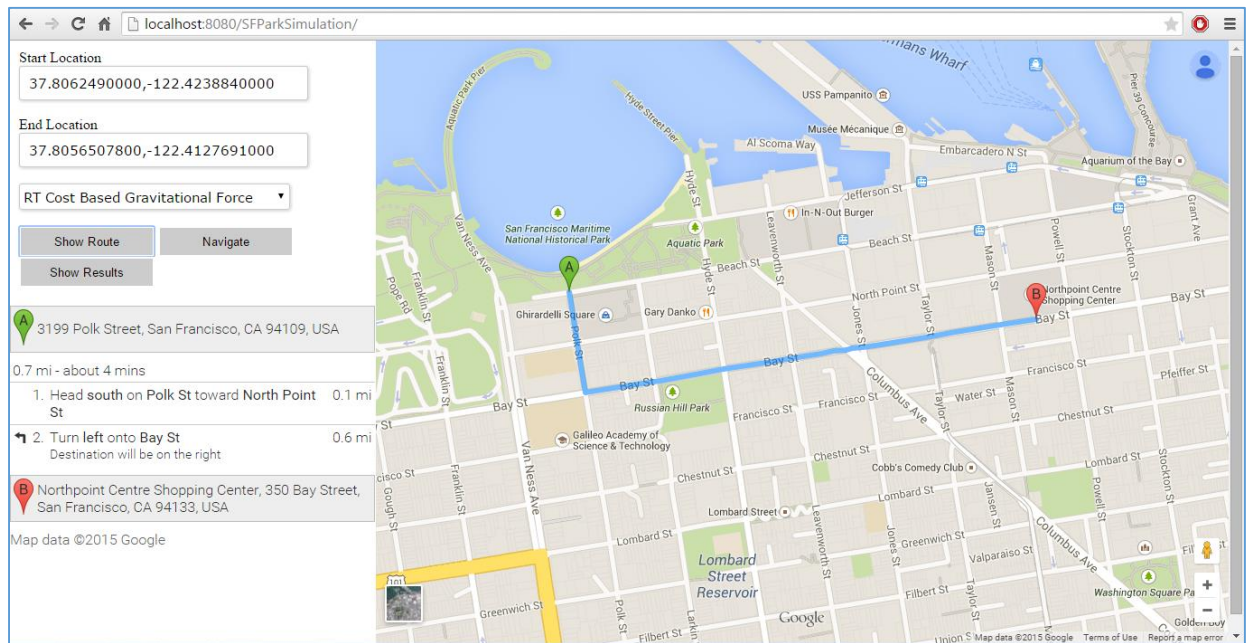
In total we had 86 valid user inputs for each algorithm, each algorithm had same user inputs including the date and time. Additionally, the date used for these 86 user inputs were spread across the entire week thus covering each day of the week. Similarly, the time used was spread across the entire day. 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 from 0% to 70%.

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

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 2 shows a snapshot from the website of study. As we can see in Figure 2, 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.

Below are the assumptions used for the web based study:

1. The driving speed of the user: 15 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.

# 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 study 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:

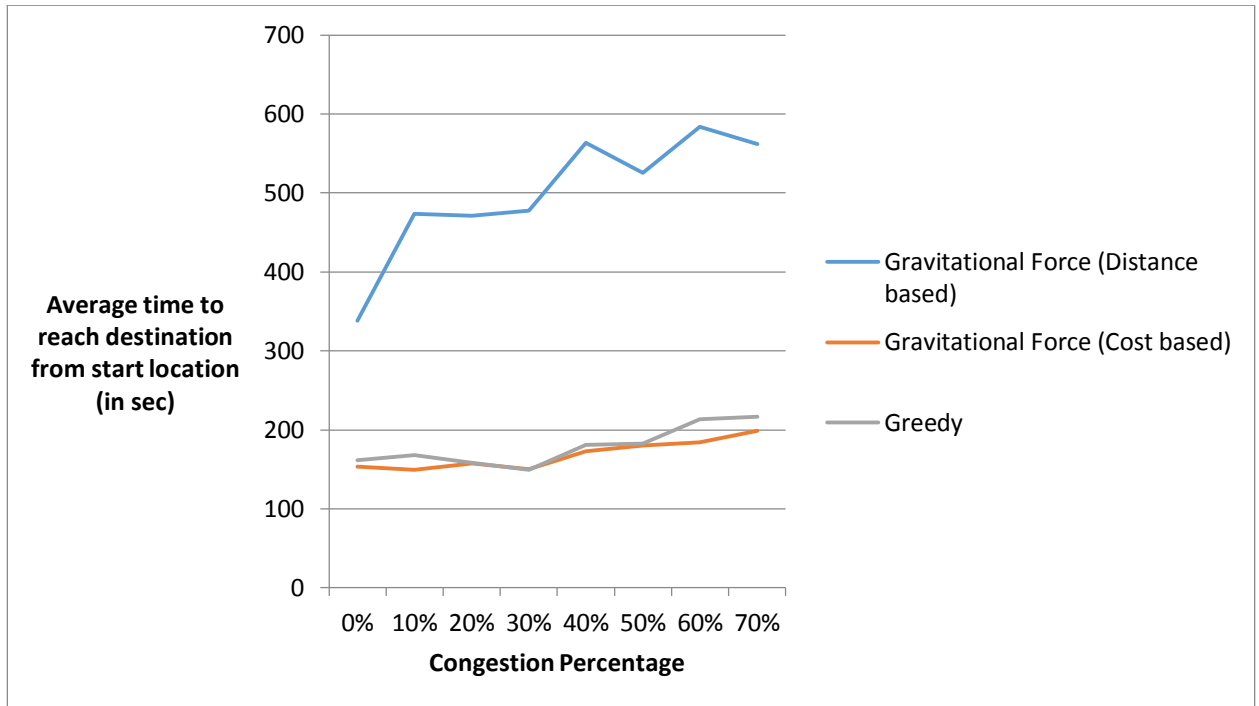


FIGURE 3: DETERMINISTIC APPROACH: METRIC 1

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). 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. Clearly, distance based gravitational algorithm has shown a poor performance compared to the other two algorithm as it does not accounts User's destination while allotting a parking block. Additionally, the performance is bound to degrade as the congestion increases as the available parking slots have decreased.

2. Cost based Gravitational force has shown the best performance among all three algorithms which is closely followed by greedy algorithm.
3. 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. Unlike cost based gravitational algorithm, 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 155 seconds between 0-30 percent of congestion and then for congestion percent above 30 both the algorithms starts 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 200 seconds for 70% congestion.

## METRIC 2:

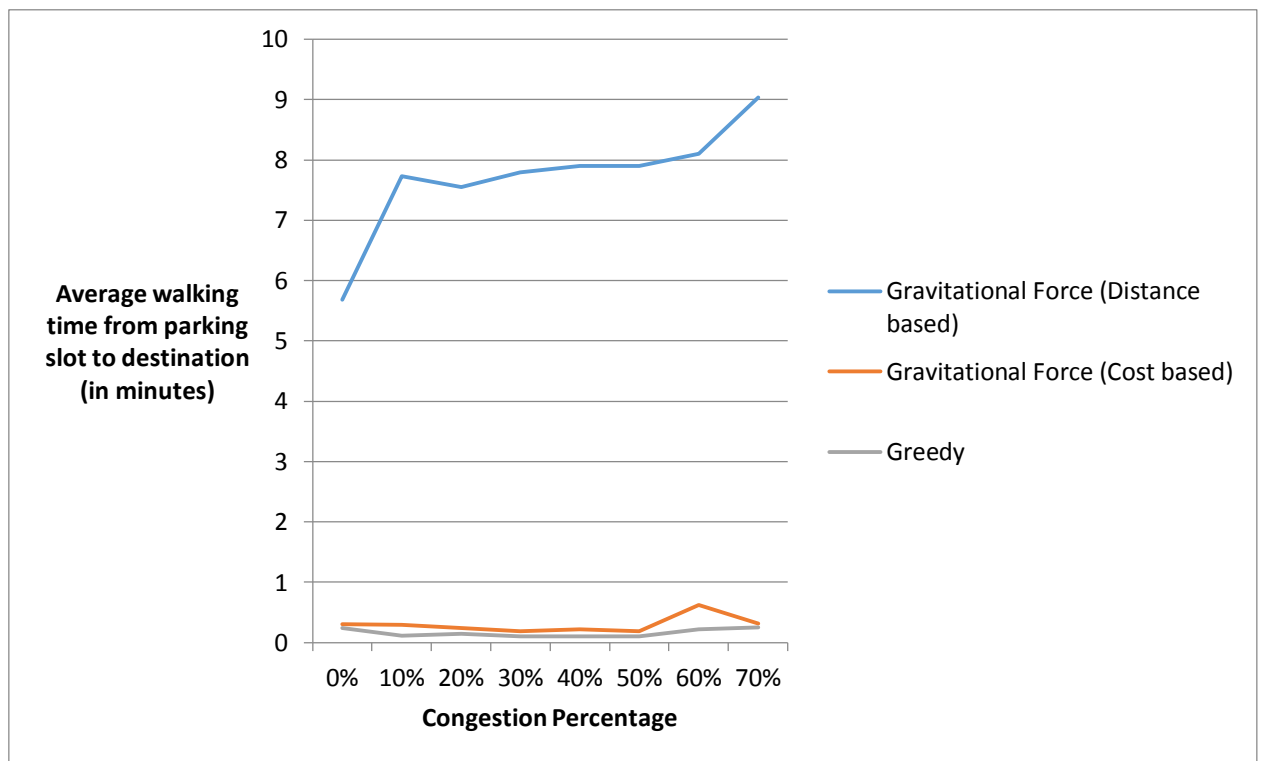
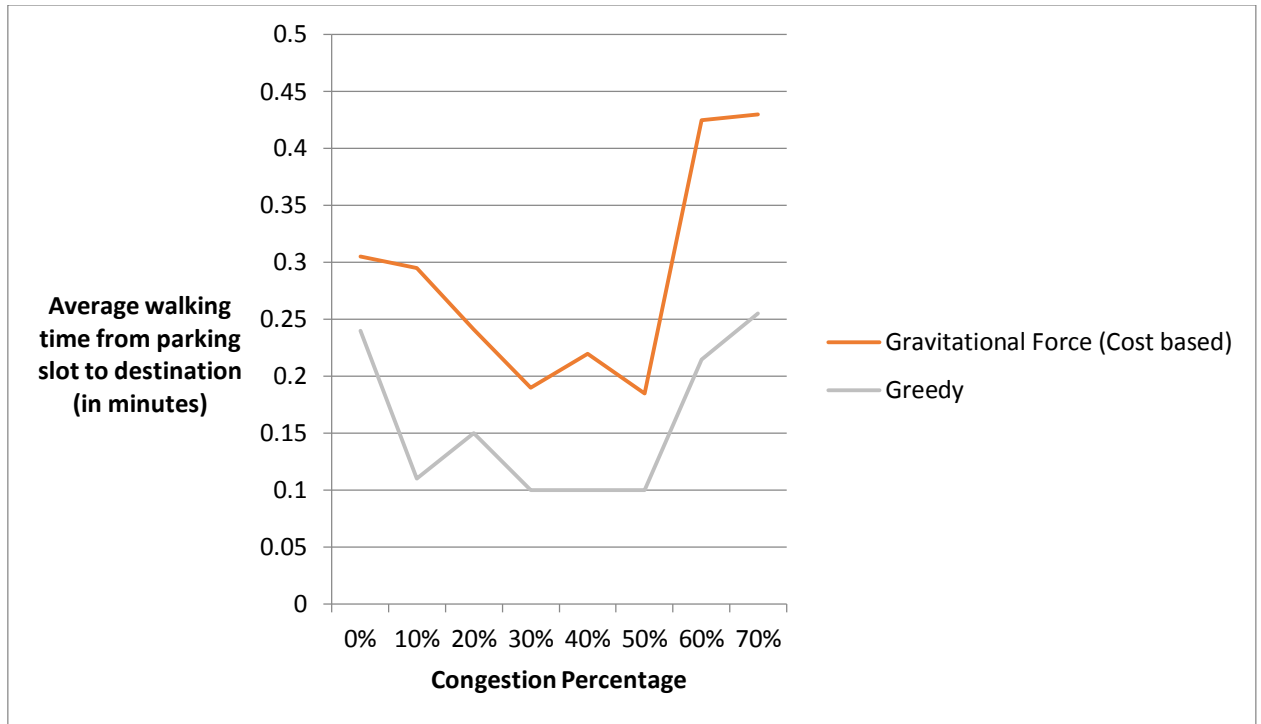


FIGURE 4: DETERMINISTIC APPROACH: METRIC 2



**FIGURE 5: DETERMINISTIC APPROACH: GRAVITATIONAL & GREEDY: METRIC 2**

The above figures 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 walking time from allotted parking slot to user's destination (in minutes).

#### **Observations and Justification:**

1. Distance based gravitational algorithm shows a poor performance compared to other two algorithm as it does not accounts the walking time from the allotted parking block to the user destination. Thus, we can see that the average walking time for the user is almost 8 minutes across all the congestion levels
2. From Figure 4 we can see that cost based gravitational and greedy performance almost the same, but if we look at the graphs closely, as shown in Figure 5, we can see that greedy performs well as it accounts user's destination and disregards the possibility of a miss or re-route.
3. Thus, cost based gravitational algorithm accounts the possibility of a miss or re-route and performs really well compared to greedy algorithm in metrics 1 but at the same time it trades the walking time (a small amount of time) compared to greedy algorithm.

## Probabilistic Approach

### METRIC 1:

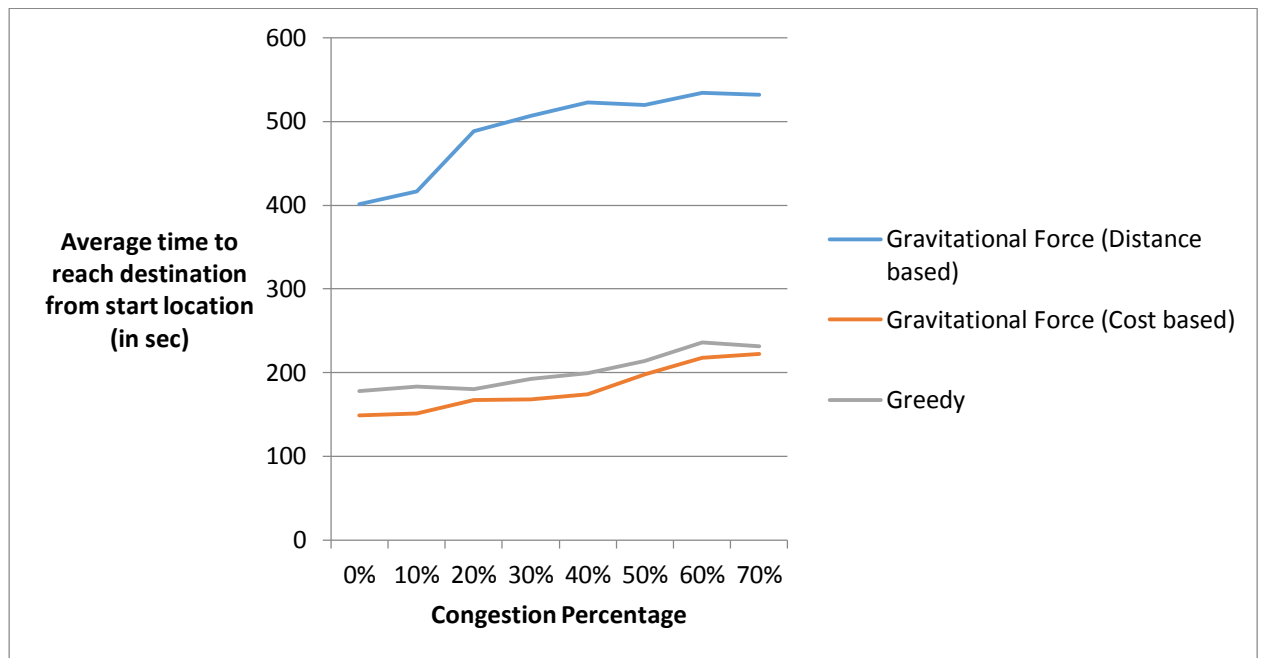


FIGURE 6: PROBABILISTIC APPROACH: METRIC 1

The Figure 6 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). 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 discussed above for deterministic approach, the distance based algorithm shows a poor performance compared to other algorithms as it does not accounts user's destination. Additionally, the performance is bound to degrade as the congestion increases as the available parking slots have decreased.
2. As discusses above for deterministic approach, the cost based gravitational and greedy shows the same kind of performance only when 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 221 seconds for both these algorithms with congestion level of 70%.

## METRIC 2:

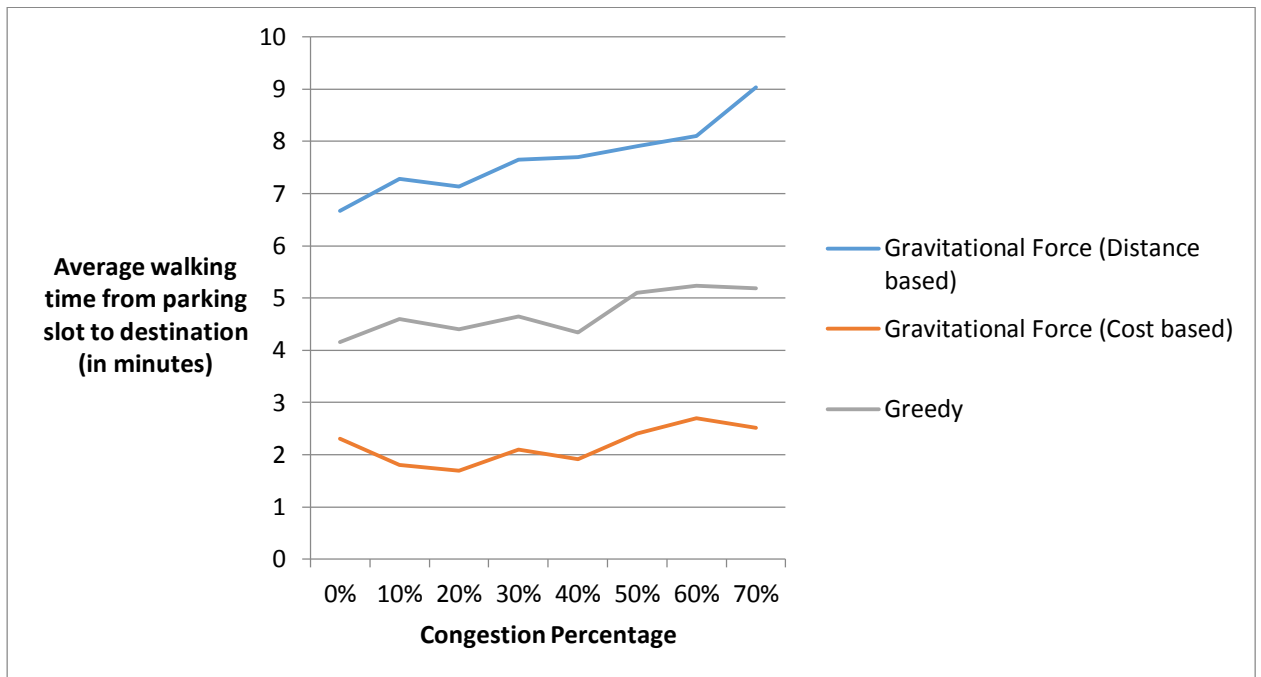


FIGURE 7: PROBABILISTIC APPROACH: METRIC 2

The above Figure 7 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 walking time from allotted parking slot to user's destination (in minutes).

### Observations and Justification:

1. As discussed for previous category, distance based gravitational shows poor performance compared to other two algorithms in this category with an average walking time near to 8 minutes.
2. Interesting thing to note here is that cost based gravitational has shown a better performance than greedy, unlike deterministic approach, the reason is because the greedy algorithm did not account the possibility of getting a parking slot rather we can say that not considering the reroutes has degraded the performance of greedy algorithm with a probabilistic data set as the walking time has increased compared to earlier discussed approach.
3. Distance based approach has shown relatively better performance than greedy algorithm with probabilistic data set though the average walking time has increased compared to deterministic approach but considering the probabilistic nature of data the increase is bound to happen.



## Baseline along with overall performance

### METRIC 1:

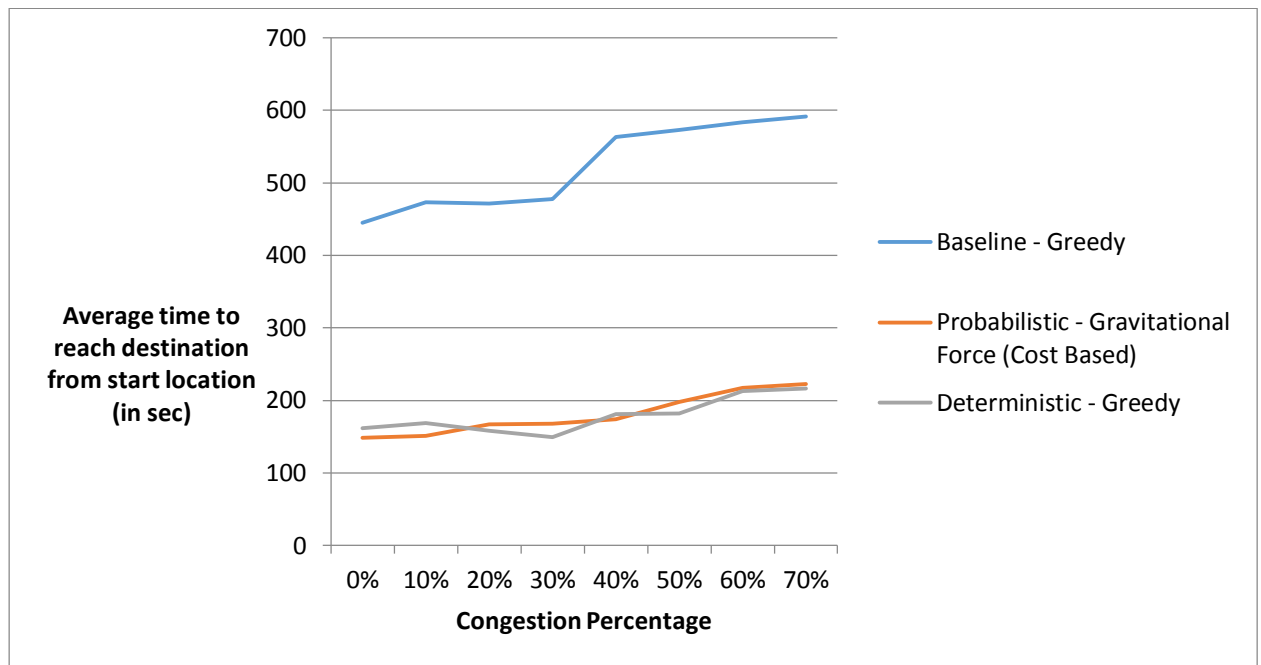


FIGURE 8: BASELINE : METRIC 1

The Figure 8 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), 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.
2. Cost based gravitational force algorithm for probabilistic approach and Greedy algorithm for deterministic approach shows nearly the same performance but definitely, deterministic approach has an edge over probabilistic approach and hence greedy algorithm for deterministic approach gives a better average time to reach to the destination over the earlier.
3. The performance degrades as the congestion increases but the interesting fact to note is that the rate at which the performance degrade is also a factor for considering the performance of the algorithm. As shown in above graph, the

performance of probabilistic approach and deterministic approach is way better than baseline approach.

## METRIC 2:

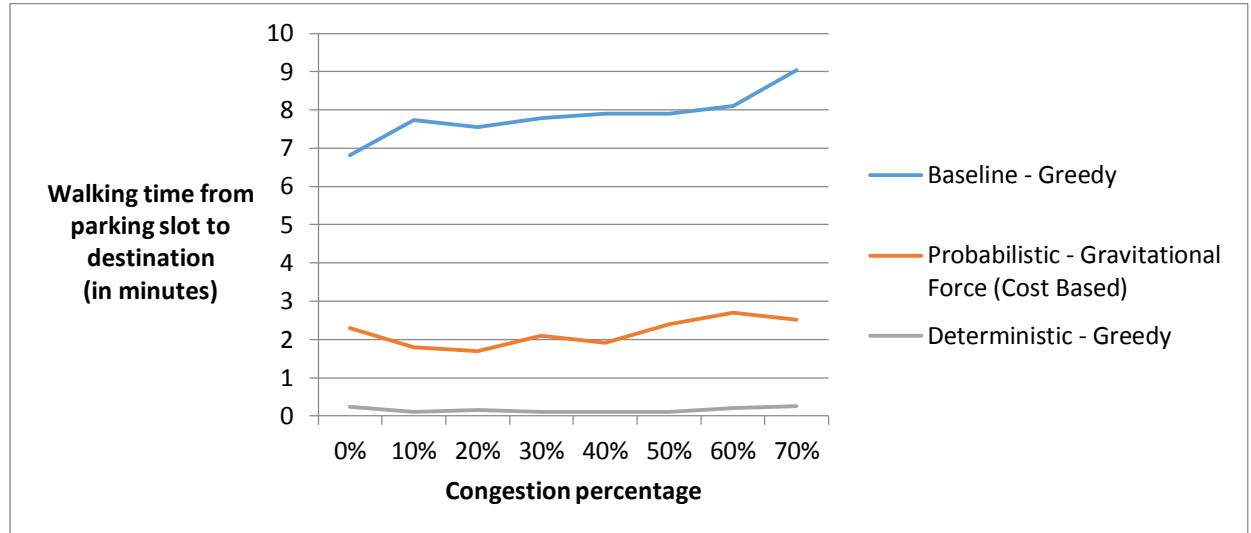


FIGURE 9: BASELINE: METRIC 2

The Figure 9 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), 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 walking time from allotted parking slot to user's destination (in minutes).

### Observations and Justification:

1. As discussed above the baseline algorithm has the worst performance of all the algorithms evaluated. The average walking time is close to 8 minutes.
2. Comparing the probabilistic approach and real time approach, it is not hard to conclude that the deterministic approach performs well as the data set that is considered is real time and hence the algorithms can avoid re-routes and find a parking location as close to the user's destination which is less likely in probabilistic approach.
3. Above graph shows the difference in the performance of both the approaches, the average walking time for a deterministic approach is about half a minute whereas the average walking time for probabilistic approach is about 2 minutes.
4. Also, the deviation of average walking time in deterministic approach is relatively low compared to that of probabilistic approach.

# Functional Design

## Design

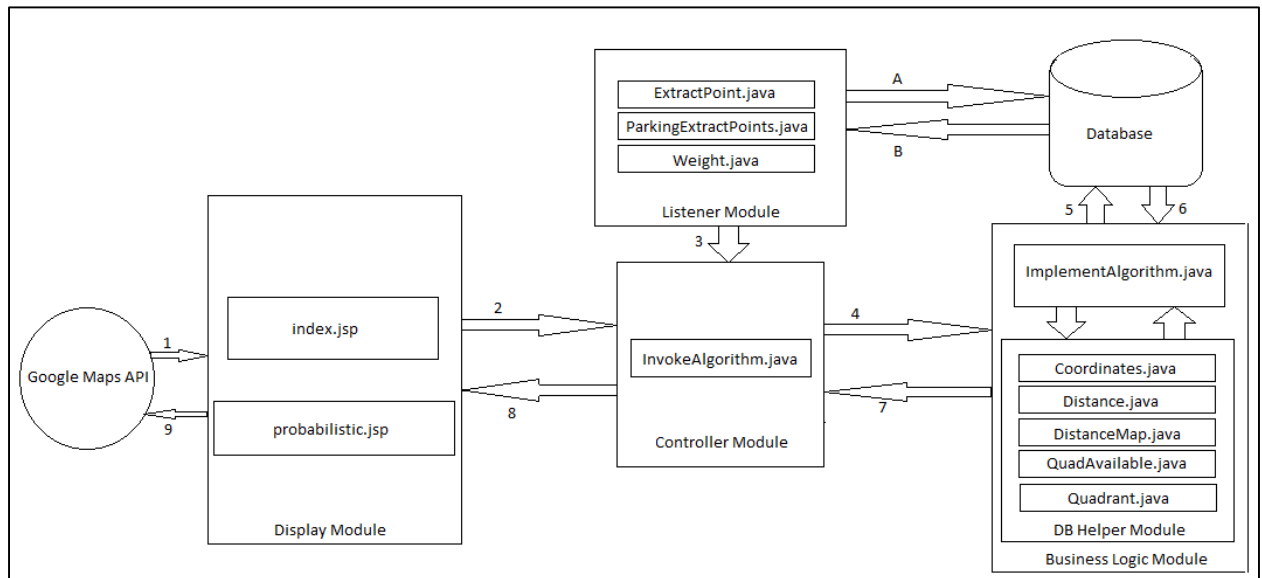


FIGURE 10: APPLICATION DESIGN

### 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 5 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. Display Module

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

- index.jsp: Displaying the UI for real time algorithm implementation
- Probabilistic.jsp: Displaying the UI for probabilistic and baseline algorithm implementation
- 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 Module

- Controller Module consists of a servlet InvokeAlgorithm.java
- Controller module 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 module 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 Listener Module.
- Also the approximation of intended user destination B to the nearest parking block midpoint is done using Listener Module

## 4. Implementation Module

- This is the most important module and this is the module where algorithm executes
- This module consists of
  - ImplementAlgorithm.java: Java class that executes all the algorithms
  - DBHelper Module: Consists of various helper classes that help ImplementAlgorithm.java in implementing the algorithm

- This module takes as input A or C, B and algorithm name and then executes the algorithm by querying the Database and using DBHelper classes
- This module returns the co-ordinates of the allocated P to Controller.

## 5. Listener Module

- 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 ImplementAlgorithm 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
  - ExtractPoints: extract all nodes in the database
  - ParkingExtractPoints: these points are basically the mid-points of all the parking blocks
  - Weight: 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 index.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 a 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. DBHelper 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
9. 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

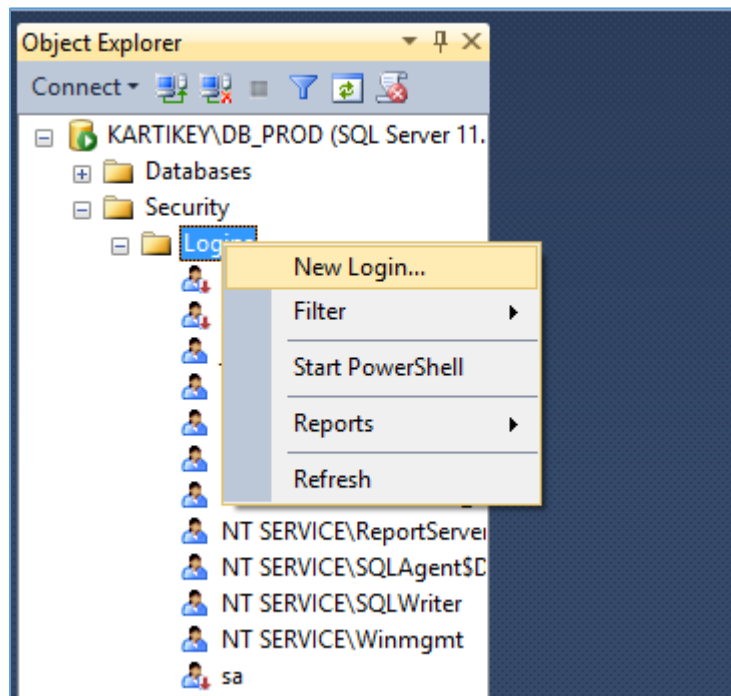
Following software are needed for running the application:

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

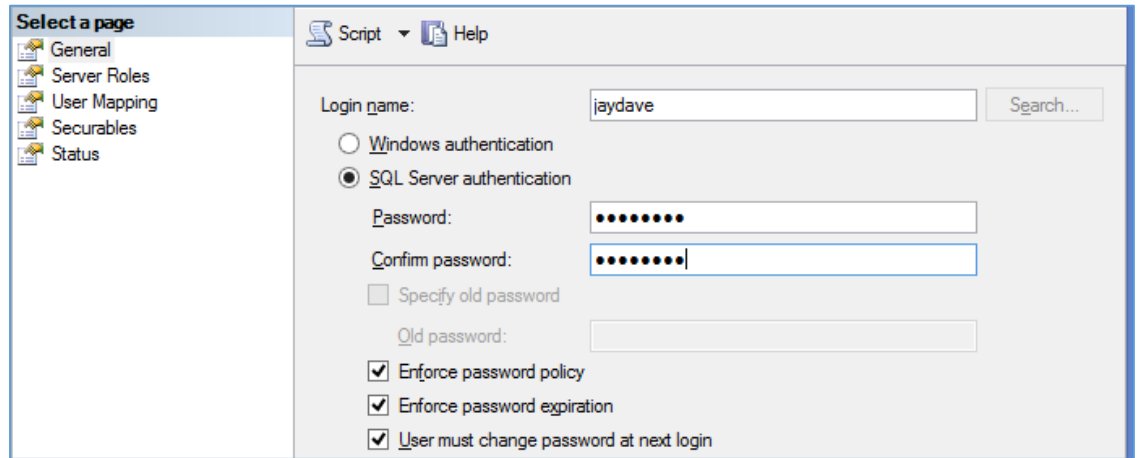
## Configuring SQL Server 2012 with Tomcat 8.0

Steps for configuring SQL Server 2012 with Tomcat 8.0

- a. Create initial database configuration for a super user by following the instruction guided while installing the SQL server 2012.
- b. Copy the database file (SFPark111) 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 “SFPark”
- 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: “jaydave” and password: “jay12345” and check all boxes as depicted in below diagram and then click on “OK”



The screenshot shows the 'Select a page' dialog box in SQL Server Enterprise Manager. The 'General' page is selected. The 'Login name' field contains 'jaydave'. The 'Authentication' section has 'SQL Server authentication' selected. The 'Password' and 'Confirm password' fields are filled with dots. The 'Specify old password' checkbox is unchecked. The 'Old password' field is empty. The 'Enforce password policy', 'Enforce password expiration', and 'User must change password at next login' checkboxes are all checked.

- l. Refresh security and then right click on the user “jaydave” and give the admin rights to this user.
- m. Database is now setup.

## Running Simulations

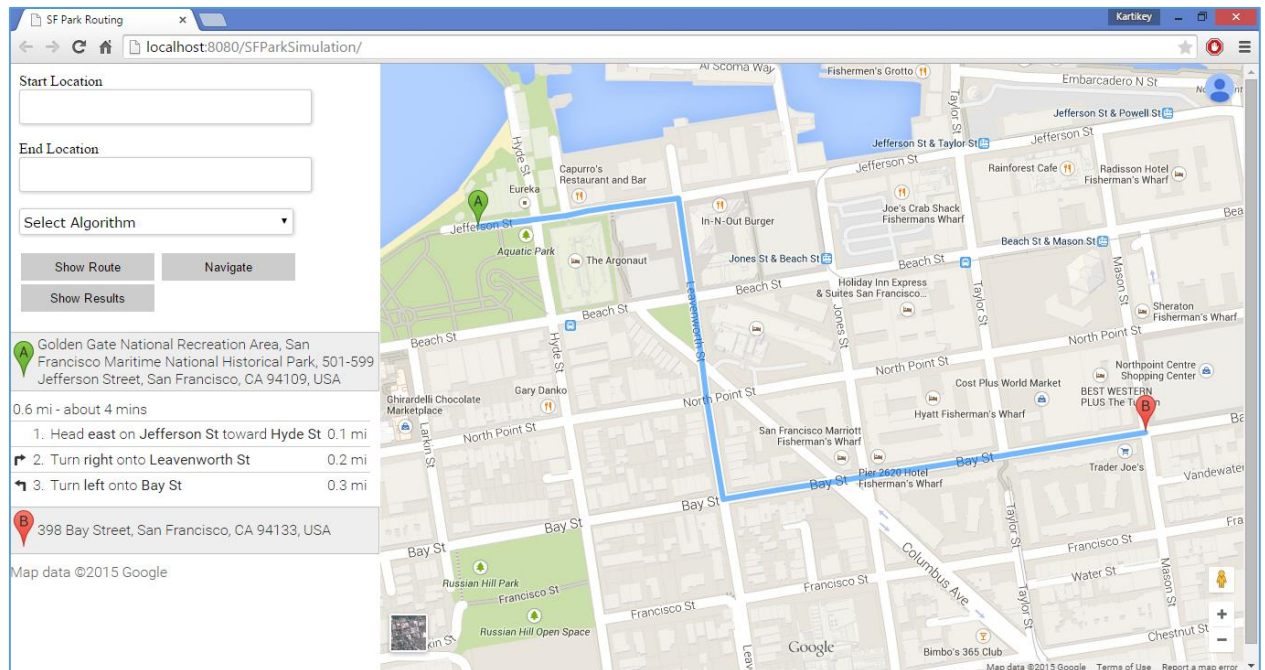
Steps for running the application

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 SFParkSimulation.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/SFParkSimulation>
8. For accessing the probabilistic and baseline algorithm use the link <http://localhost:8080/SFParkSimulation/Probabilistic.jsp>

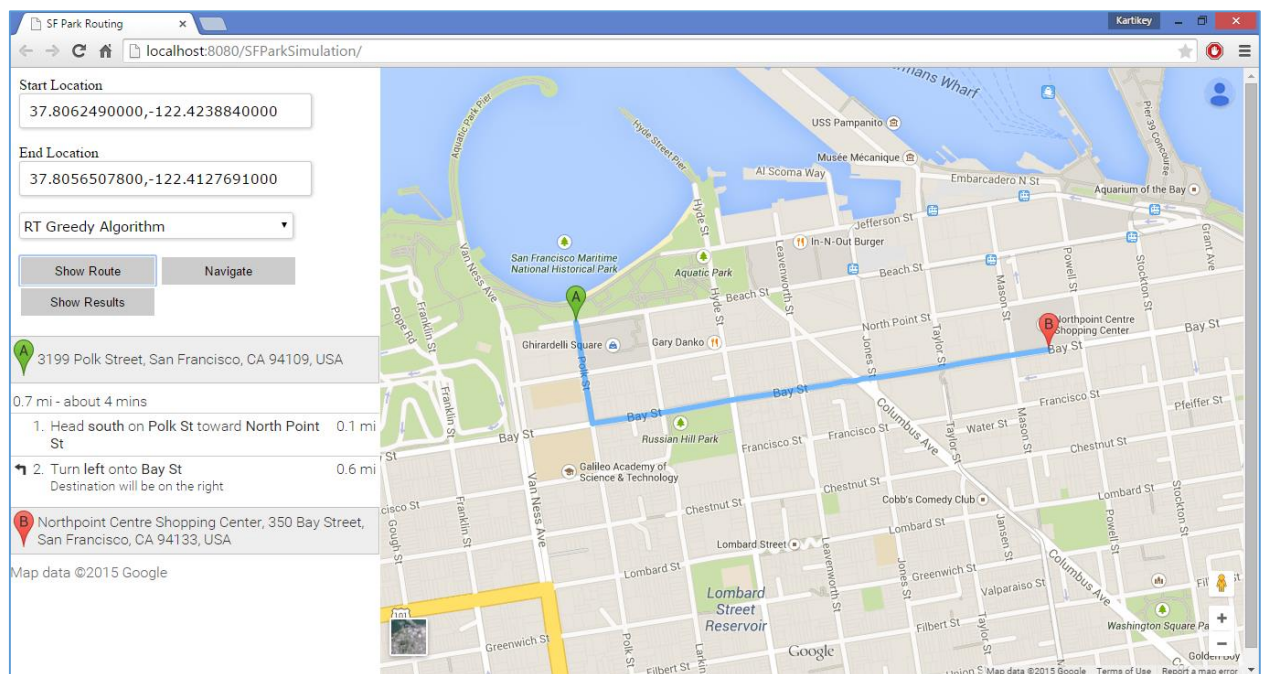
## Sample Run

1. Use the localhost <http://localhost:8080/SFParkSimulation> to launch the application on a web browser:





2. Enter the co-ordinates, select desired algorithm and then click on “Show Route” followed by “Navigate” to start the execution of the algorithm



3. Once the user reaches the destination, then click on show “Show Results” to get the user final location and the time elapsed

SF Park Routing

localhost:8080/SFParkSimulation/

Start Location

37.8062490000,-122.4238840000

End Location

37.8056507800,-122.4127691000

RT Greedy Algorithm

Show Route

Navigate

Show Results

A

Northpoint Centre Shopping Center, 350 Bay Street, San Francisco, CA 94133, USA

1 ft - about 1 min

1. Head on Bay St 1 ft

B

Northpoint Centre Shopping Center, 350 Bay Street, San Francisco, CA 94133, USA

Map data ©2015 Google

Start Location: 37.8062490000,-122.4238840000

End Location: 37.8056507800,-122.4127691000

Final Location: 37.80565078,-122.4127691

Slot acquired in 94.58725599999889 seconds

RadioShack

Safeway

Walgreens

24 Hour Fitness

Payment Alliance International, Inc

Chase Bank

Bay St

Bay St

Bay St

Project Zen Massage & Bodywork

DeLise dessert cafe

Google

Map data ©2015 Google

Terms of Use

Report a map error