**DATABASE MANAGEMENT SYSTEMS (CS 581)**

**SEARCH OF SPATIO – TEMPORAL RESOURCES**

**FINAL PROJECT REPORT**

**GROUP 1**

MINU FRANCIS

POOJA DONEKAL

SHVETHA SUVARNA

SURBHI ARORA

**Table of Contents**

Contents

[1. Introduction 5](#_Toc449964228)

[2. Motivation 5](#_Toc449964229)

[3. Goal 5](#_Toc449964230)

[4. Dataset 6](#_Toc449964231)

[4.1 Provided Files 6](#_Toc449964232)

[4.2 Data Plot 6](#_Toc449964233)

[5. System Architecture 7](#_Toc449964234)

[5.1. Overview 7](#_Toc449964235)

[5.2. User Interface of the Web Application 7](#_Toc449964236)

[5.3. System Architecture 8](#_Toc449964237)

[5.3.1. User Interface 8](#_Toc449964238)

[5.3.2. Business Logic 8](#_Toc449964239)

[5.3.3. Data Access Module 8](#_Toc449964240)

[5.3.4. Database 9](#_Toc449964241)

[5.3.5. External Interface 9](#_Toc449964242)

[5.3.6. Google Maps API 9](#_Toc449964243)

[6. Methodologies 9](#_Toc449964244)

[6.1. Level of Information - Real Time Parking Information is provided 9](#_Toc449964245)

[6.1.1. Problem Description 9](#_Toc449964246)

[6.1.2. Holistic Architecture 9](#_Toc449964247)

[6.1.3. Pre-Analysis of given data 10](#_Toc449964248)

[6.1.4. Alternate Methods Considered 11](#_Toc449964249)

[6.1.5. Method Chosen - Parking Slot Availability determination based on the Gravitational Pull Algorithm 11](#_Toc449964250)

[6.2 Level of Information – Historical Data Regarding Parking is provided 21](#_Toc449964251)

[6.2.1. Problem Description 21](#_Toc449964252)

[6.2.2. Holistic Architecture 22](#_Toc449964253)

[6.2.3. Pre-Analysis of Given Data 22](#_Toc449964254)

[6.2.4. Alternate Methods Considered 23](#_Toc449964255)

[6.2.5. Method Chosen: Parking Slot Availability estimation based on Weighted Average 23](#_Toc449964256)

[6.3. Level of Information - Parking with no data about the parking slots availability. 34](#_Toc449964257)

[6.3.1. Problem Description 34](#_Toc449964258)

[6.3.2. Holistic Architecture 34](#_Toc449964259)

[6.3.3. Alternate Methods Considered 35](#_Toc449964260)

[6.3.4. Method Chosen - Parking Slot Availability estimation based on nearest neighbor to the source location, in a region having pre-defined boundaries. 35](#_Toc449964261)

[7. Comparison of algorithms 45](#_Toc449964262)

[8. Technologies Used 46](#_Toc449964263)

[9. Steps to Run the Project 46](#_Toc449964264)

[10. Acknowledgment 46](#_Toc449964265)

[References 47](#_Toc449964266)

**LIST OF FIGURES**

[Figure 1 Visualization of the Intersections Provided 6](#_Toc449964267)

[Figure 2 User Interface of the Web Application 7](#_Toc449964268)

[Figure 3 System Architecture 8](#_Toc449964269)

[Figure 4 Holistic Architecture 10](#_Toc449964270)

[Figure 5 Screenshot of Pre cached Data 11](#_Toc449964271)

[Figure 6 Simulation of Gravitational Pull Algorithm 12](#_Toc449964272)

[Figure 7 Flow Chart for GPA 13](#_Toc449964273)

[Figure 8 Test Results for Gravitational Pull Algorithm with 0% Congestion Level 15](#_Toc449964274)

[Figure 9 Test Results for Gravitational Pull Algorithm with 10% Congestion Level 16](#_Toc449964275)

[Figure 10 Test Results for Gravitational Pull Algorithm with 20% Congestion Level 16](#_Toc449964276)

[Figure 11 Test Results for Gravitational Pull Algorithm with 30% Congestion Level 17](#_Toc449964277)

[Figure 12 Test Results for Gravitational Pull Algorithm with 40% Congestion Level 17](#_Toc449964278)

[Figure 13 Test Results for Gravitational Pull Algorithm with 50% Congestion Level 18](#_Toc449964279)

[Figure 14 Plot between ‘Time to find Parking’ and ‘Congestion Levels’ 19](#_Toc449964280)

[Figure 15 Plot between ‘Query Time’ and ‘Congestion Level’ for GPA 20](#_Toc449964281)

[Figure 16 Graph Comparing GPA and Greedy Approach to find Parking using Real Time Data 21](#_Toc449964282)

[Figure 17 Holistic Architecture for Algorithm based on Historic Data 22](#_Toc449964283)

[Figure 18 Pre - Cached Data: Weighted Average based on Day and Time(per hour) 24](#_Toc449964284)

[Figure 19 Flow Chart for algorithm implemented using Historic Data 25](#_Toc449964285)

[Figure 20 Screenshot of Test Data used for ecperiments 26](#_Toc449964286)

[Figure 21 Test Results for Historic Data Based Algorithm with 0% Congestion 28](#_Toc449964287)

[Figure 22 Test Results for Historic Data Based Algorithm with 10% Congestion 28](#_Toc449964288)

[Figure 23 Test Results for Historic Data Based Algorithm with 20% Congestion 29](#_Toc449964289)

[Figure 24 Test Results for Historic Data Based Algorithm with 30% Congestion 29](#_Toc449964290)

[Figure 25 Test Results for Historic Data Based Algorithm with 40% Congestion 30](#_Toc449964291)

[Figure 26 Test Results for Historic Data Based Algorithm with 50% Congestion 30](#_Toc449964292)

[Figure 27 Graph between 'Time Taken by Algorithm' and 'Congestion Percentage' 31](#_Toc449964293)

[Figure 28 Graph between 'Total Time Taken by Algorithm' and 'Congestion Percentage' 32](#_Toc449964294)

[Figure 29 Graph between 'Time taken to Find Parking' and 'Congestion Percentage' 33](#_Toc449964295)

[Figure 30 Output of the Parking Search on the Web application 34](#_Toc449964296)

[Figure 31 Holistic Architecture for Uninformed Search 34](#_Toc449964297)

[Figure 32 Simulation of Uninformed Search Algorithm 36](#_Toc449964298)

[Figure 33 Flow Chart for the Uninformed Search Algorithm 37](#_Toc449964299)

[Figure 34 Test Results for Uninformed Search with 0% Congestion 39](#_Toc449964300)

[Figure 35 Test Results for Uninformed Search with 10% Congestion 39](#_Toc449964301)

[Figure 36 Test Results for Uninformed Search with 20% Congestion 40](#_Toc449964302)

[Figure 37 Test Results for Uninformed Search with 30% Congestion 40](#_Toc449964303)

[Figure 38 Test Results for Uninformed Search with 40% Congestion 41](#_Toc449964304)

[Figure 39 Test Results for Uninformed Search with 50% Congestion 41](#_Toc449964305)

[Figure 40 Graph between 'Time to Find Parking' and 'Congestion Levels' 42](#_Toc449964306)

[Figure 41 Graph between Query Time and Congestion Levels 43](#_Toc449964307)

[Figure 42 Output of the Uninformed Search on Web Application 44](#_Toc449964308)

[Figure 43 Graph showing the Comparison between various algorithms implemented 45](#_Toc449964309)

**LIST OF TABLES**

[Table 1 Dataset Provided 6](#_Toc449964310)

[Table 2 Alternate Algorithms Considered 11](#_Toc449964311)

[Table 3 Alternate Methods Considered for implementing Historical Data Based Algorithm 23](#_Toc449964312)

[Table 4 Pre cached Data for Historic Data based algorithm 23](#_Toc449964313)

[Table 5 Alternate Methods Considered for Uninformed Search 35](#_Toc449964314)

**Search of Spatio-Temporal Resources**

# Introduction

In major cities it has been observed that 30% of the traffic comprises of individuals cruising for a parking spot. This shows that the search for traffic is an enormous source of congestion. There are a number of projects whose sole aim is reduce this problem. One instance taken into consideration is the SFPark project. The SFPark is a project initiated by the San Francisco Municipal Transportation Agency, in which wireless sensors were embedded under the pavement in order to provide real time data. This application provides a list of available parking spots and the user would manually have to choose one. This would make it difficult for the user to make a decision while driving. In order to solve this issue we aim to create an algorithm that provides directions to the users at each intersection, from the start location to the destination.

# Motivation

In order to reduce the contribution of individuals looking for a parking spot to the regular flow of traffic, a method needs to be devised in order to direct the user to a place where they would find a parking slot closer to their destination. The destination should kept in mind in order to take into consideration, the distance the user has to walk from their parking slot to their destination.

# Goal

The goal of this project is to try to address the issue of parking by evaluating various algorithms for various instances of the problem. This project will consider three instances of Spatio-Temporal data and run various algorithms on these instances. This first instance will consider the given data as real time data and run the Real Time based algorithm on it. The second instance will consider the data as historical data and run the appropriate algorithm whereas the third instance will be completely uninformed. This will be then evaluated against certain metrics than can help us judge the performance of one algorithm against the other.

# Dataset

## 4.1 Provided Files

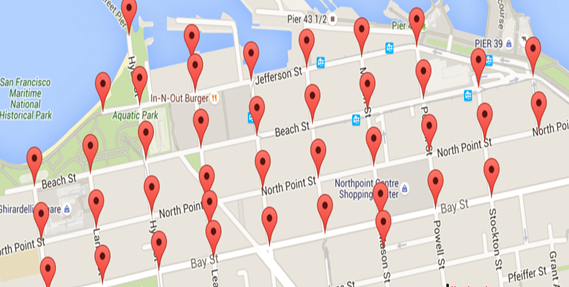
The dataset provided is from the San Francisco’s tourist area. There are 40 nodes and 63 edges in this road network. It consists of three separate CSV files containing the following data.

|  |  |
| --- | --- |
| **Data** | **Description** |
| SFPark\_nodes\_FishermansWharf.csv | Contains node data |
| SFPark\_edges\_FishermansWharf2.csv | Contains edge data |
| dbProjection\_4\_6\_12-5\_5\_12.csv | Contains the list of available parking slots |

**Table 1 Dataset Provided**

## 4.2 Data Plot

The diagram is a plot of all the possible intersections in the given data. This diagram was used a reference to visualize the various intersections while testing.



**Figure 1 Visualization of the Intersections Provided**

# System Architecture

## Overview

The system is implemented in the form of a web based application which takes parking requests from the user, performs the necessary calculations and directs the users at each intersection until they find a parking slot.

## User Interface of the Web Application

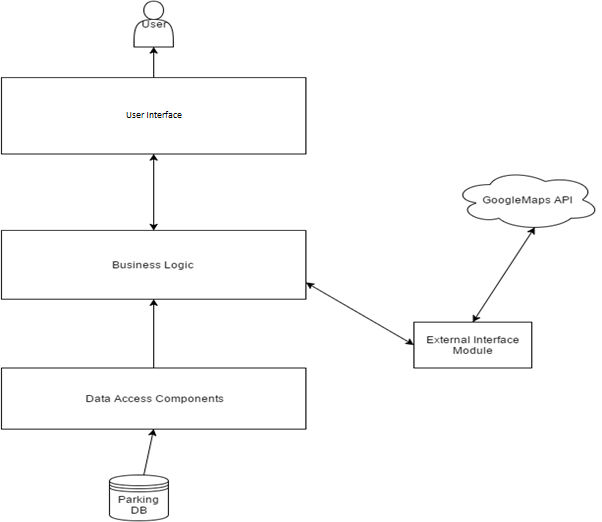
Below is the screen shot of the web application the user will interface with.



**Figure 2 User Interface of the Web Application**

## System Architecture

Below is the overall architecture for the system.

****

**Figure 3 System Architecture**

The description of the individual modules are as follows:

### 5.3.1. User Interface

The User Interface Module provides a web based interface for the user to interact with the system. This interface takes input parameters for location and time. Based on the values passed, it interacts with the business logic to provide the necessary output. This is the only means for the user to communicate with the application.

### 5.3.2. Business Logic

This component is responsible for making computations based on the algorithm used. This is the heart of the entire system. It interacts with the Data Access component and all other external interfaces to fetch data .The final output from this module is sent to the User Interface module.

### 5.3.3. Data Access Module

This module directly interfaces with the database. The necessary queries are run in this module are provided as an output to the Business layer.

### 5.3.4. Database

This layer is populated with the provided and pre-cached data .It stores all the information relevant to the application.

### 5.3.5. External Interface

This layers acts as an interface for the application to interact with external REST Services. The data obtained from the REST services is fed into the application from this layer.

### Google Maps API

This layer is an external interface which provides the necessary output as a response to the application’s request. The output from this module is processed by the business layer and is displayed to the user from the User Interface module.

# Methodologies

Each of algorithms for the individual tasks will be covered under this section.

## 6.1. Level of Information - Real Time Parking Information is provided

### 6.1.1. Problem Description

Assuming you lived in San Francisco and had access to the real time data from SFpark. In this task, we are devising an algorithm(s) which guides a user to find a street parking spot. We are using Gravitational Pull of a parking spot which is Analogous to physics. We can define the gravitational force of a parking block b towards a vehicle v as

|  |
| --- |
| ***F(v, b) = nb / cost(v, b)2*** |

***nb*** : The number of available parking spots at block b.

**Cost**: The time taken by the vehicle to reach the parking spot plus time taken by the user to walk back[1].

Average Driving Speed Taken: 20 miles/hour

Average Walking Time Per Hour: 3 miles/hour

### 6.1.2. Holistic Architecture

This architecture represents the modified version of the system architecture in order to suit the needs of the algorithm used in task 1.

C:\Users\minuf\Downloads\real_data_arch.png

**Figure 4 Holistic Architecture**

### 6.1.3. Pre-Analysis of given data

For any input, the algorithm should be able to determine the availability of the parking spot at that location, for the time of the current request based on its current availability. It should also consider the effect of all the parking slots that can be reached by taking a certain turn at an intersection.

### 6.1.4. Alternate Methods Considered

Some of the methods that were taken into consideration before coming up with the solution were as follows.

|  |  |  |  |
| --- | --- | --- | --- |
| **No** | **Method Name** | **Method Description** | **Drawback** |
| 1 | Greedy Approach – Max Slot | The block containing the maximum number of parking slots will be chosen as “Optimal” by the algorithm. | There might be a number of optimal parking slots which are close to the user’s location and this technique fails to take that into consideration. |
| 2 | Greedy Approach – Closest Slot | The block closest to the user will be chosen and the user will be redirected to that particular slot | Since the closest slot will be chosen by the algorithm cannot guarantee the availability of the slot once the user reaches the location. This might lead to the algorithm redirecting to different parking spots multiple times. |

**Table 2 Alternate Algorithms Considered**

### 6.1.5. Method Chosen - Parking Slot Availability determination based on the Gravitational Pull Algorithm

#### Pre-Cached data

The following data is pre-cached in order to enable the algorithm to perform efficiently at runtime. We will pre-cache the distances from and to the nodes and its neighbors in order to reduce the calculation overhead.

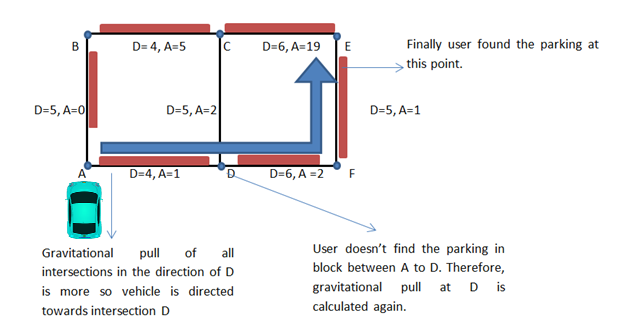
The data pre-cached here is the distance from each of the intersections to its neighbors.

#### 

**Figure 5 Screenshot of Pre cached Data**

The distance between neighbors is pre computed and stored in the form of a JSON Object where the key is the intersection ID and the value is the distance between two intersections.

#### Algorithm



**Figure 6 Simulation of Gravitational Pull Algorithm**

1. The input taken by the algorithm will be the user’s current location and the parking request time.
2. If the user is not at an intersection, he first will be directed to the closest intersection.
3. The algorithm will use the pre-computed data to fetch the neighbors and distance from the user’s location.
4. Based on the values fetched the gravitational pull formulae will be applied to all the intersection.

***F(v, b) = nb / cost(v, b)2***

1. Based on the value obtained from this formula the user will be sent in the direction with the maximum gravitational pull.
2. If the user does not find a parking slot, the computations will be performed once again using current intersection as the source and the current time stamp as the request time.
3. If the user location is matched to latitude, longitude of the available parking slot at that time, parking is found and the algorithm will terminate.

#### Flowchart

Below is the flow chart representing the steps to implement algorithm to find parking spot using Gravitational Pull formula.

****

**Figure 7 Flow Chart for GPA**

#### Individual Experiments

The Experiments were conducted based on the following metrics

* Time taken for the algorithm to find a parking spot under various levels of congestion
* Time taken for the user to find a parking spot under various levels of congestion

#### Experiment Inputs

The test data is stored in a file named “TestData.xls”. The csv file contains the following data corresponding to 225 different source – timestamp combinations.

1. Latitude – latitude of the initial position of user

2. Longitude – longitude of the initial position of user

3. Timestamp – Time at which user is searching for parking

#### Experiment Methodology

In this case, experiments are conducted by adding congestion. Congestion is induced by reducing the number of available parking slots by a certain percentage of the maximum possible value of the parking slot availability as given in the database (i.e. 19). The experiment is conducted for various levels of congestion – 10%, 20%, 30%, 40% and 50%. The test data is stored in a file named “TestData.xls”.

The csv file contains the following data:

1. Latitude – latitude of the initial position of user

2. Longitude – longitude of the initial position of user

3. Timestamp – Time at which user is searching for parking

4. Congestion Percent – The congestion percentage that is taken into consideration.

#### Experiment Steps

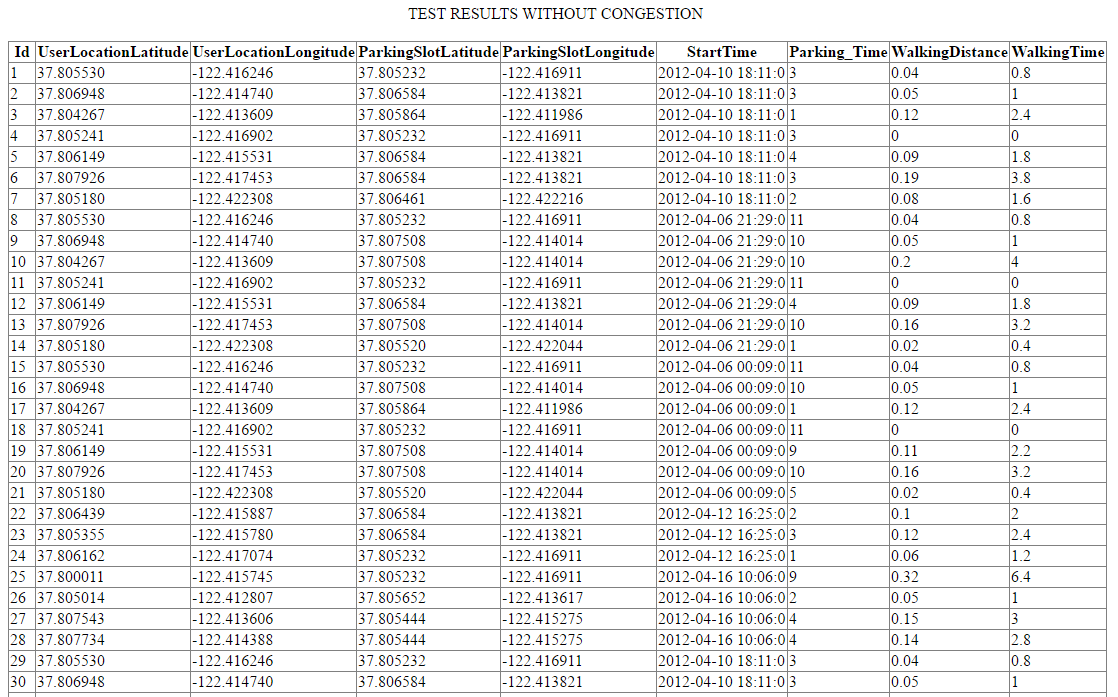
In order to find the results of the algorithm, a button named ‘Show Results’ is available. On click of this button, the data in the spreadsheet is taken as input and the algorithm is run by considering the inputs one at a time. The results of the test cases are stored in the database and displayed on the UI.

#### Experiment Results

Results of the experiments conducted for GPA are displayed below.

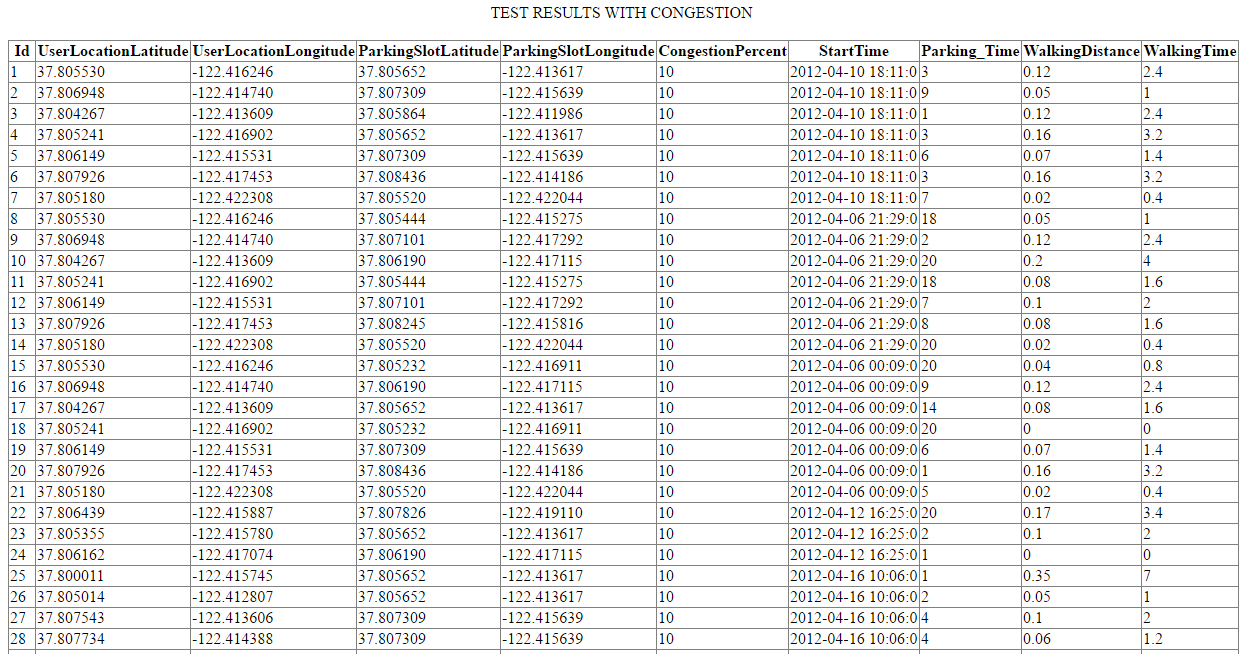
* UserLocationLatitude/UserLocationLongitude – Initial start location of User
* ParkingSlotLatitude/ParkingSlotLongitude – Location of the allotted parking block
* StartTime – Time at which user starts searching for a parking
* Parking\_Time(Minutes) – Time taken by the user to find a parking block.
* WalkingDistance(Miles) – Distance taken by the user to walk to destination from the allotted parking block
* Walking Time(Minutes) – Time taken by the user to walk to destination from the allotted parking block

##### Congestion Level – 0%



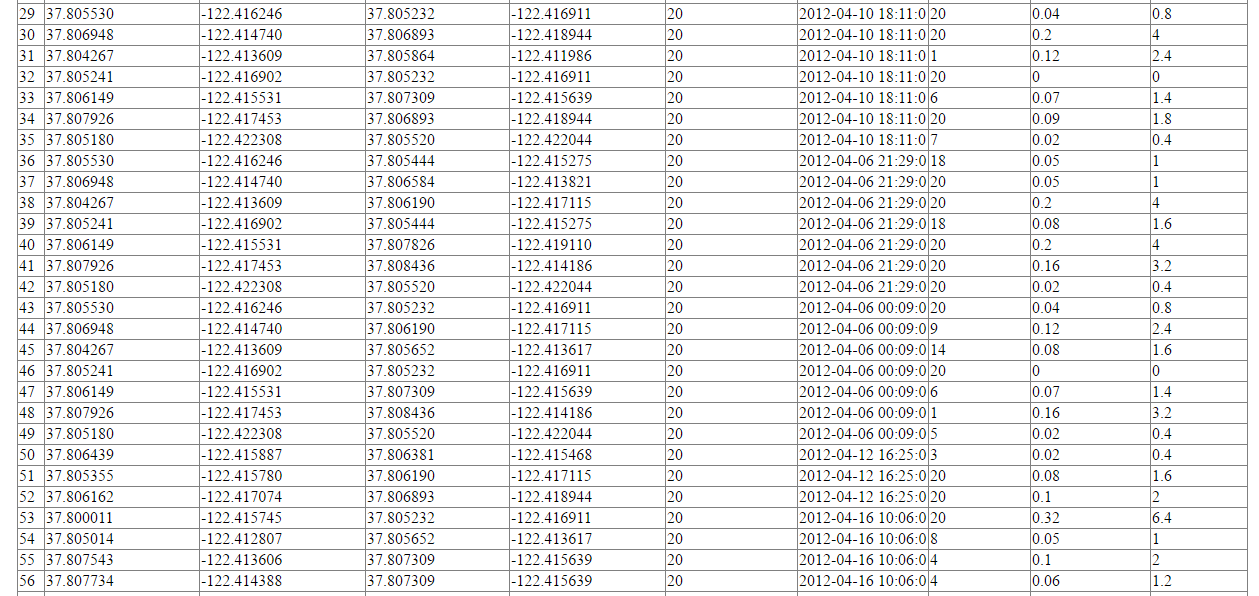
**Figure 8 Test Results for Gravitational Pull Algorithm with 0% Congestion Level**

##### Congestion Level – 10%



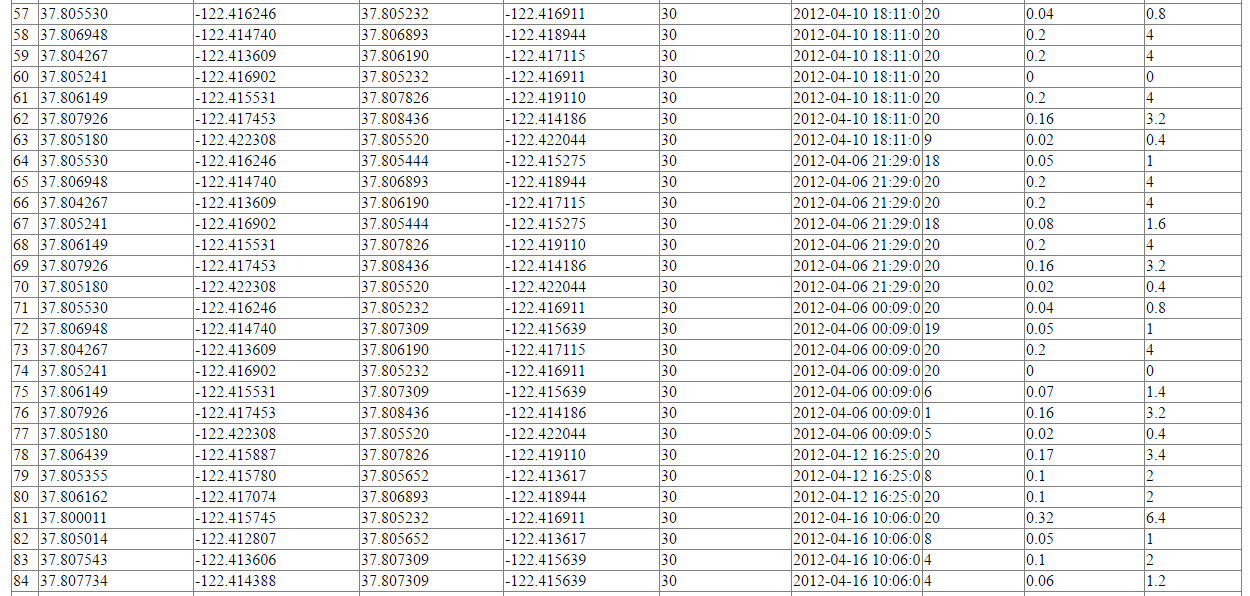
**Figure 9 Test Results for Gravitational Pull Algorithm with 10% Congestion Level**

##### Congestion Level – 20%



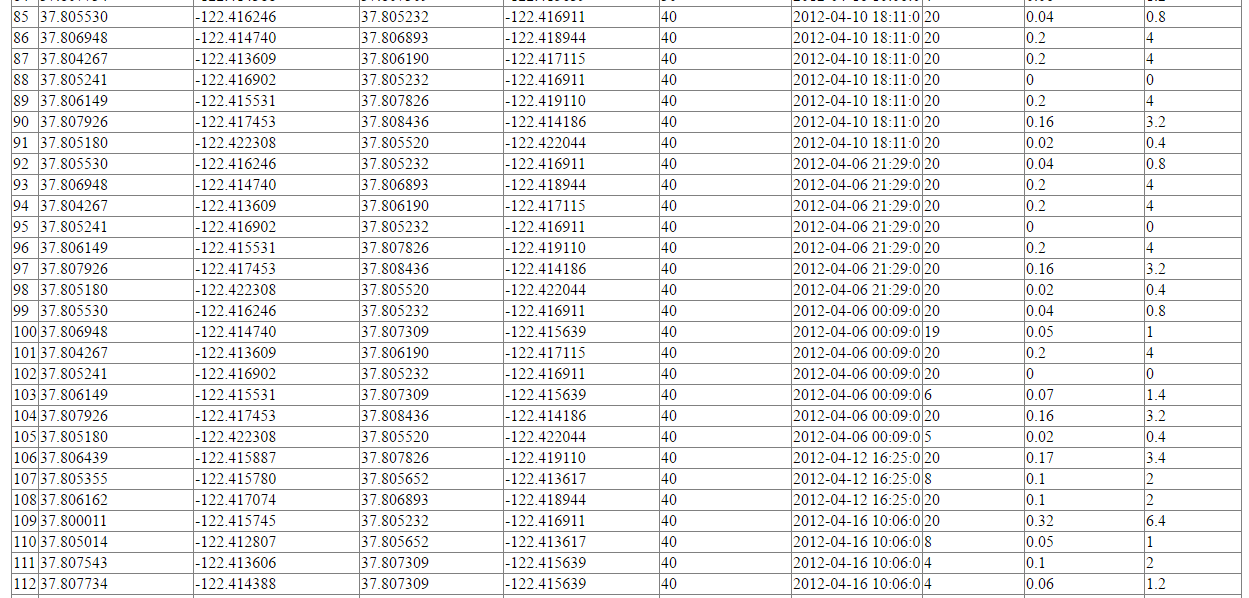
**Figure 10 Test Results for Gravitational Pull Algorithm with 20% Congestion Level**

##### Congestion Level - 30%



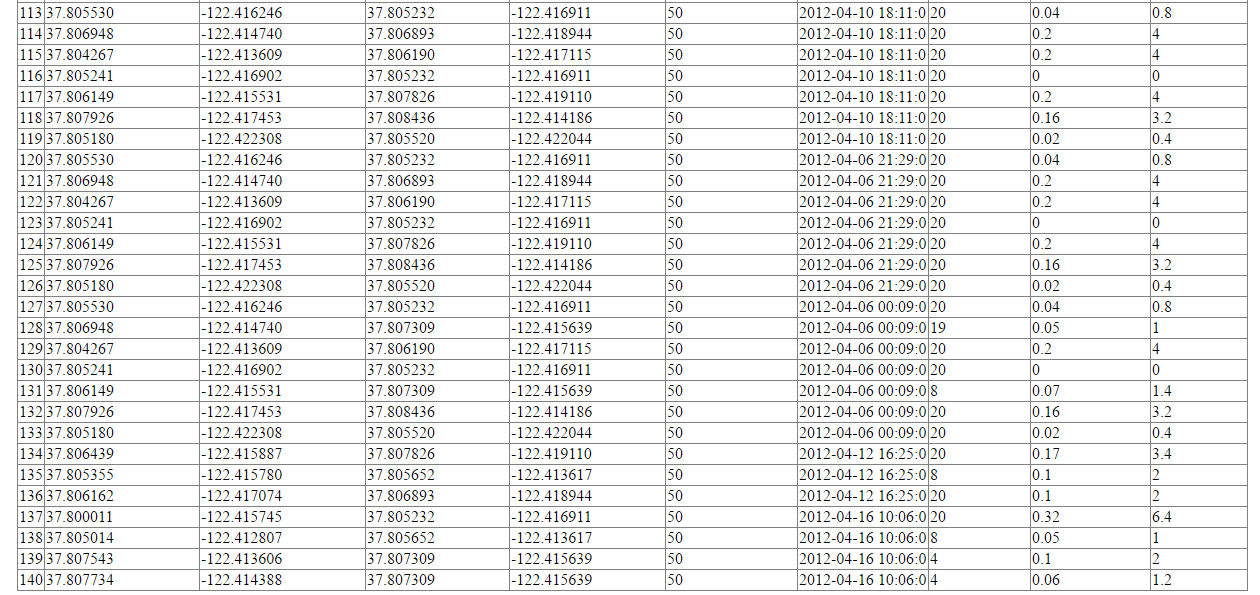
**Figure 11 Test Results for Gravitational Pull Algorithm with 30% Congestion Level**

##### Congestion Level – 40%



**Figure 12 Test Results for Gravitational Pull Algorithm with 40% Congestion Level**

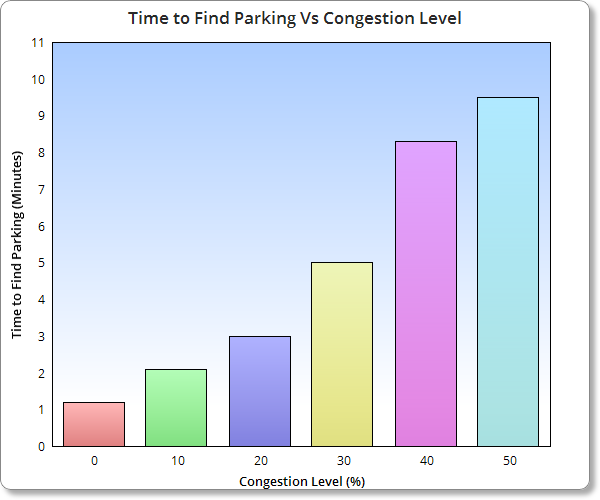
##### Congestion Level – 50%



**Figure 13 Test Results for Gravitational Pull Algorithm with 50% Congestion Level**

#### Experiment Analysis

##### Graph1: Time to find Parking Vs Congestion Levels

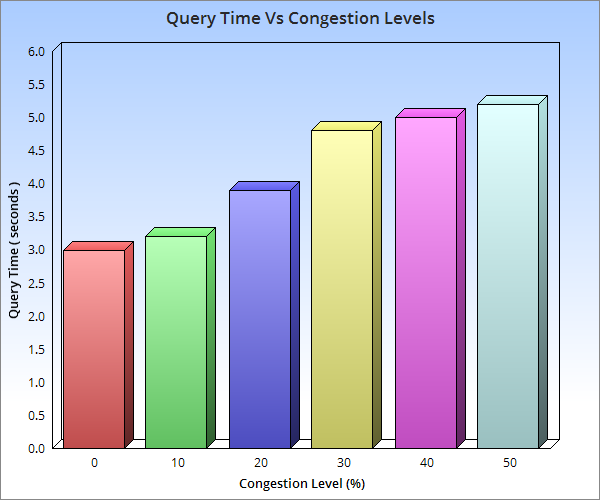
****

**Figure 14 Plot between ‘Time to find Parking’ and ‘Congestion Levels’**

##### Graph Description

The above graph shows the analysis of ‘Time taken to find parking’ at different ‘Congestion Levels’. We can conclude that the time taken to find a parking slot, on average increases with an increase in the congestion level. This happens since the number of available slots decrease. There is a drastic increase in the total time at 40% congestion. This is because most of the parking slots in the databases is less than 40% congestion level. Thus there is a drastic increase in the total time taken by the user to reach the destination by for the last two levels of congestion.

##### Graph2: Query Time Vs Congestion Levels

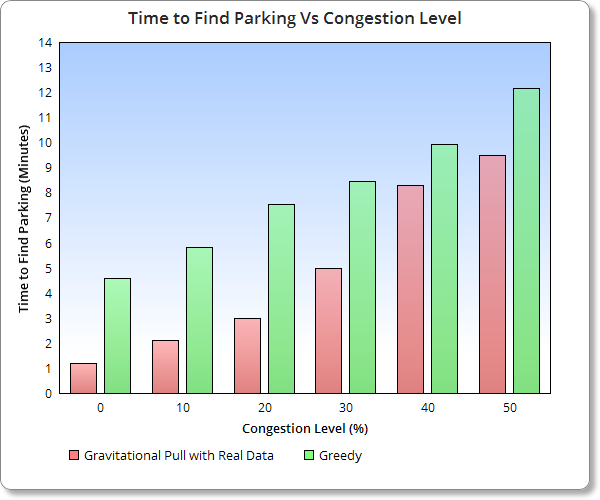
****

**Figure 15 Plot between ‘Query Time’ and ‘Congestion Level’ for GPA**

##### Graph Description:

The above graph displays a plot between ‘Query Time’ and ‘Congestion Level’. Query time refers to the running time of the algorithm and this has been recorded for various congestion levels. For each congestion level, the time plotted on the graph is the average of running time of the algorithm over 45 different inputs.

##### Graph3: Gravitational Vs Greedy



**Figure 16 Graph Comparing GPA and Greedy Approach to find Parking using Real Time Data**

##### Graph Description

As per the graph Greedy takes more time as compared to the Gravitational Pull algorithm since greedy does not consider the factors of distance and weight of the slot together. This results in poor results as compared to the gravitational pull algorithm.

## Level of Information – Historical Data Regarding Parking is provided

### 6.2.1. Problem Description

Considering the data provided as historical data we need a method that efficiently predicts the availability for the parking slot for the current Time. Metrics to approximate street parking availability information at a block level based on prior data can help reduce the expenditure incurred. This algorithm should be able to make use of the available data from three months and generate a good approximation for the availability for the parking slot for any request time and date.

### 6.2.2. Holistic Architecture

This diagram describes the modifications to be made to the general system architecture to suit the needs of Task 2.

Untitled Diagram

**Figure 17 Holistic Architecture for Algorithm based on Historic Data**

### 6.2.3. Pre-Analysis of Given Data

This step was performed to analyze as to what parts of the given data could be utilized for this task. For a random input, the algorithm should be able to determine the availability of the parking spot at that location for that particular time which should be accurate up to a minute. Since we do not have data for every single minute in the day, the time data can be clubbed together in order to determine the availability of the parking slot. The data also considers empty values and unavailable park spots. This should also be taken care of by the algorithm so as to prevent outliers while making decisions for availability.

### 6.2.4. Alternate Methods Considered

|  |  |  |  |
| --- | --- | --- | --- |
| **No.** | **Method Name** | **Method Description** | **Drawback** |
| 1 | Average based Parking Allocation | The block containing parking slots will be assigned individual weights based on the average of the availability of available parking slots for that particular hour. | The outliers cause the algorithm to produce incorrect weights, thus leading to overestimation or underestimation of the availability of the parking slot depending on the type of outlier for that block. |
| 2 | Parking Slot Weight Estimation based on Combined Day and Hour Data | This method determines the weight of the parking slot but considering the mode of availability for that particular block | Since the data for all the days and hours will be combined, it results in a drop in the overall accuracy of the algorithm. This is because the availability of some of the parking slots is highly affected by the time and the day. |

**Table 3 Alternate Methods Considered for implementing Historical Data Based Algorithm**

### Method Chosen: Parking Slot Availability estimation based on Weighted Average

#### 6.2.5.1. Pre-Cached Data

The weighted averages calculated for each of the parking blocks will be pre-calculated in order to improve efficiency during runtime. Pre-Cached Data is stored in the following format. The distances to the various blocks will be reused from Task 1.

|  |  |  |
| --- | --- | --- |
| **No.** | **Name** | **Description** |
| 1 | Block\_Id | The ID used to distinguish blocks |
| 2 | Hours (1-24) | The days of the week will be stored in a single level JSON format under each of the hours |

**Table 4 Pre cached Data for Historic Data based algorithm**



**Figure 18 Pre - Cached Data: Weighted Average based on Day and Time(per hour)**

The weighted average will be pre-calculated using the following formulae. This will keep the outliers in check and will give the best approximation of the availability for the parking slot.

|  |
| --- |
| (Availability of Parking Slot \* Number of Occurrences of the availability of that parking slot )/Sum of availabilities |

#### 6.2.5.2. Pre-Cached Data Generation Code

The code to generate this data is in the CreateCache.aspx and CreateCache.aspx.cs file.

#### Algorithm

1. The input taken by the algorithm will be the user’s current location and the parking request time.
2. Once the user’s current location is obtained the algorithm will fetch the weighted average from the precomputed values for that particular request time.
3. Based on the precomputed values for distance and weighted average the below formulae will be applied to the retrieved data.

|  |
| --- |
| *(Weighted Avg) / (distance)* |

1. Based on the weights assigned to the parking slots, the algorithm will fetch the path from the user’s location to the slot picked by the algorithm using the Google Maps API.
2. The algorithm will obtain a set of points that are intersections which fall on that route.
3. As the user reaches the first intersection on the path the algorithm will re-compute the availabilities for that new location based on the new time.
4. The user will be then redirected to the new location.
5. The algorithm will run recursively till the user finds a parking slot.
6. It will terminate once the user reaches their destination.

#### Flow Chart

Below is the flow chart for the Algorithm implemented based on historical data

C:\Users\minuf\Downloads\flow (4).png

**Figure 19 Flow Chart for algorithm implemented using Historic Data**

#### Individual Experiments

The Experiments were conducted based on the following metrics

* Time taken for the algorithm to find a parking spot under various levels of congestion.
* Time taken for the user to find a parking spot under various levels of congestion

#### Experiment Inputs

The Algorithm is fed with 225 input values each varying in the following.

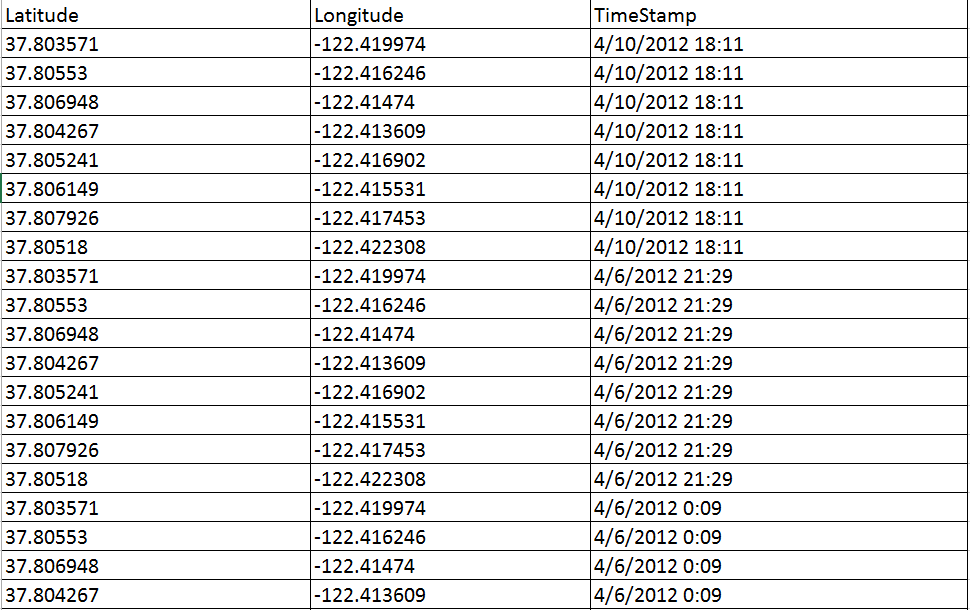
* Timestamp
* Start Location – Latitude and Longitude of the location

The inputs consist of 50% randomly generated data and 50% specifically varied inputs. This was done to make sure that most of the possible testing conditions are covered and the measure of accuracy and efficiency would represent the quality of the algorithm.

#### Experiment Methodology

The algorithm will be fed with the following Test Data.

Below is a part of the test data



**Figure 20 Screenshot of Test Data used for ecperiments**

In order to measure the accuracy of the algorithm we will fetch the time taken by the user to reach his destination for various levels of congestion. Congestion is simulated by reducing the availability of the blocks by a particular level. The algorithm is tested for 10%, 20%, 30%, 40% and 50% Congestion levels. The Test Data is in the form of a CSV File (Test Data.csv).

The Test Data is in the following format.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| SourceAddress | Latitude | Longitude | TimeStamp | CongestionPercent |

This will be fed to the function calculating the route for the user.

#### Experiment Steps

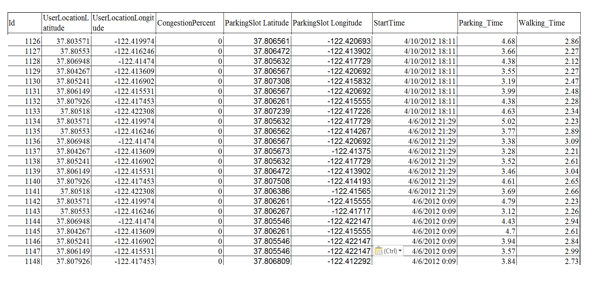
1. The Test File should be in the format mentioned above.
2. On the Main Home Page “HistoricMainPage.aspx” choose the location of the Test File.
3. Click the “Show Results” button.
4. The Test Cases will be run for Various Levels of Congestion.

#### Experiment Results

Results of the experiments conducted for Historical Data based Search are displayed below.

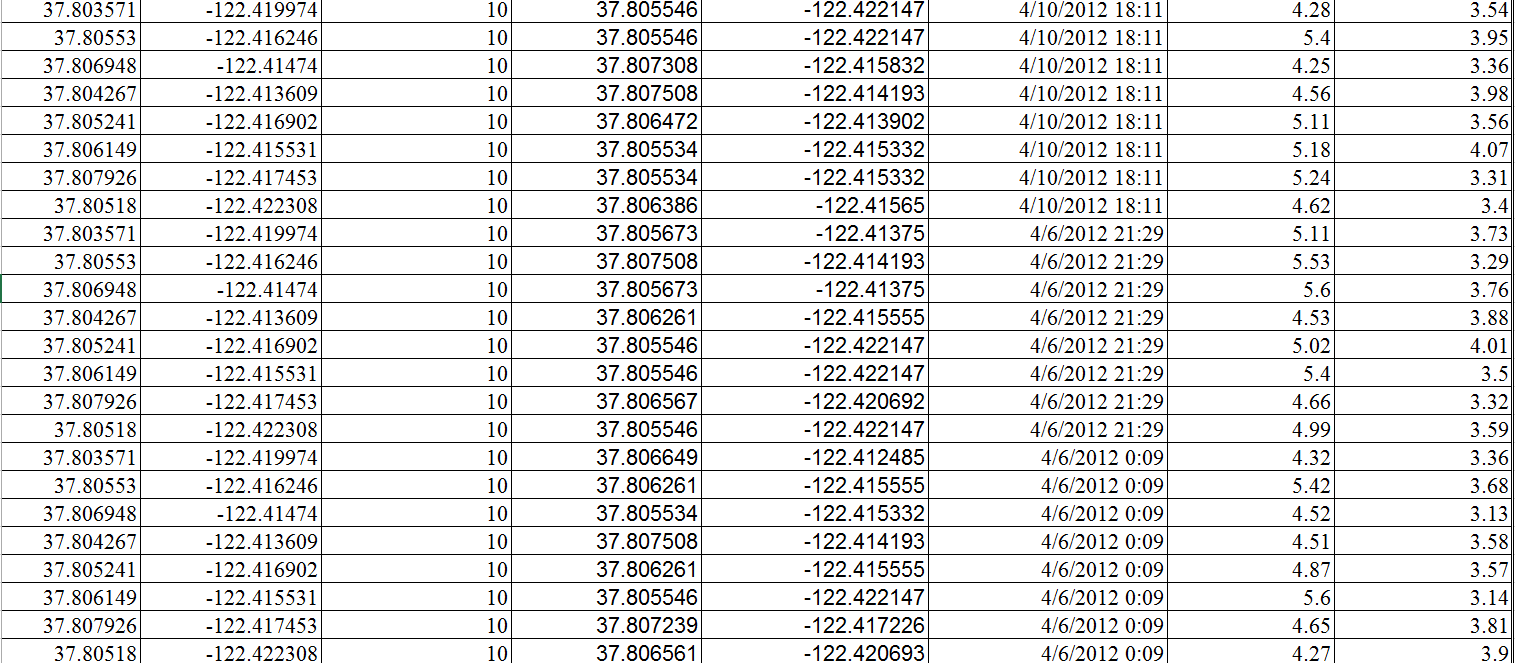
* UserLocationLatitude/UserLocationLongitude – Initial start location of User
* ParkingSlotLatitude/ParkingSlotLongitude – Location of the allotted parking block
* StartTime – Time at which user starts searching for a parking
* Parking\_Time(Minutes) – Time taken by the user to find a parking block.
* WalkingDistance(Miles) – Distance taken by the user to walk to destination from the allotted parking block
* Walking Time(Minutes) – Time taken by the user to walk to destination from the allotted parking block

##### 6.2.5.8.1 Congestion Level – 0%



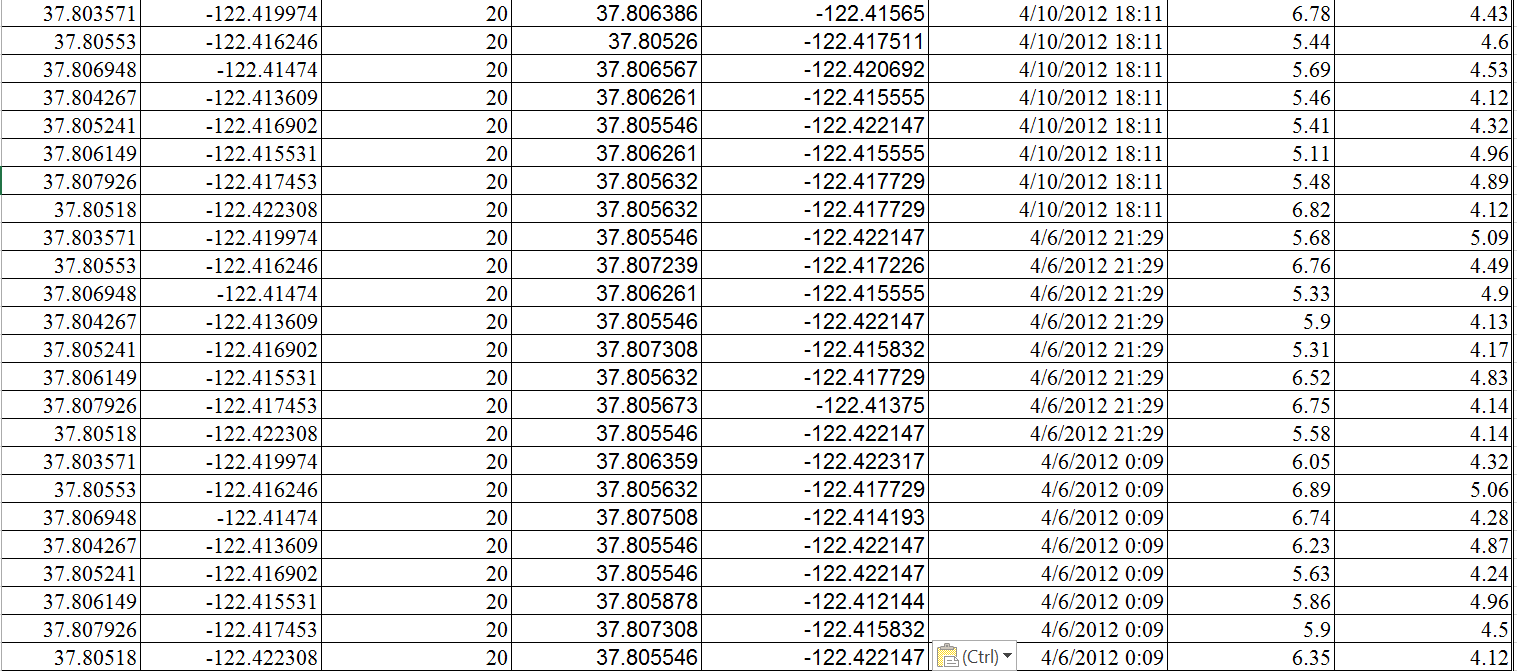
**Figure 21 Test Results for Historic Data Based Algorithm with 0% Congestion**

##### 6.2.5.8.2 Congestion Level – 10%

****

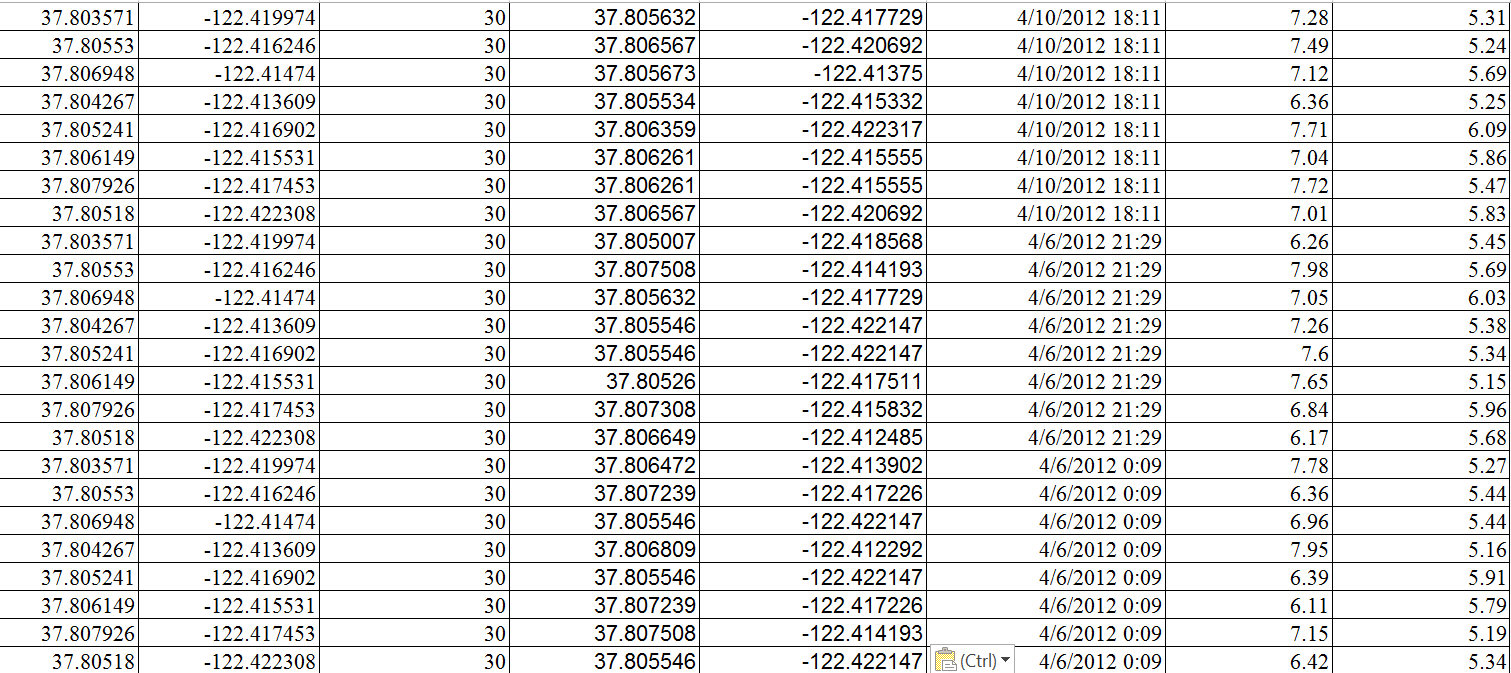
**Figure 22 Test Results for Historic Data Based Algorithm with 10% Congestion**

##### 6.2.5.8.3 Congestion Level – 20%

****

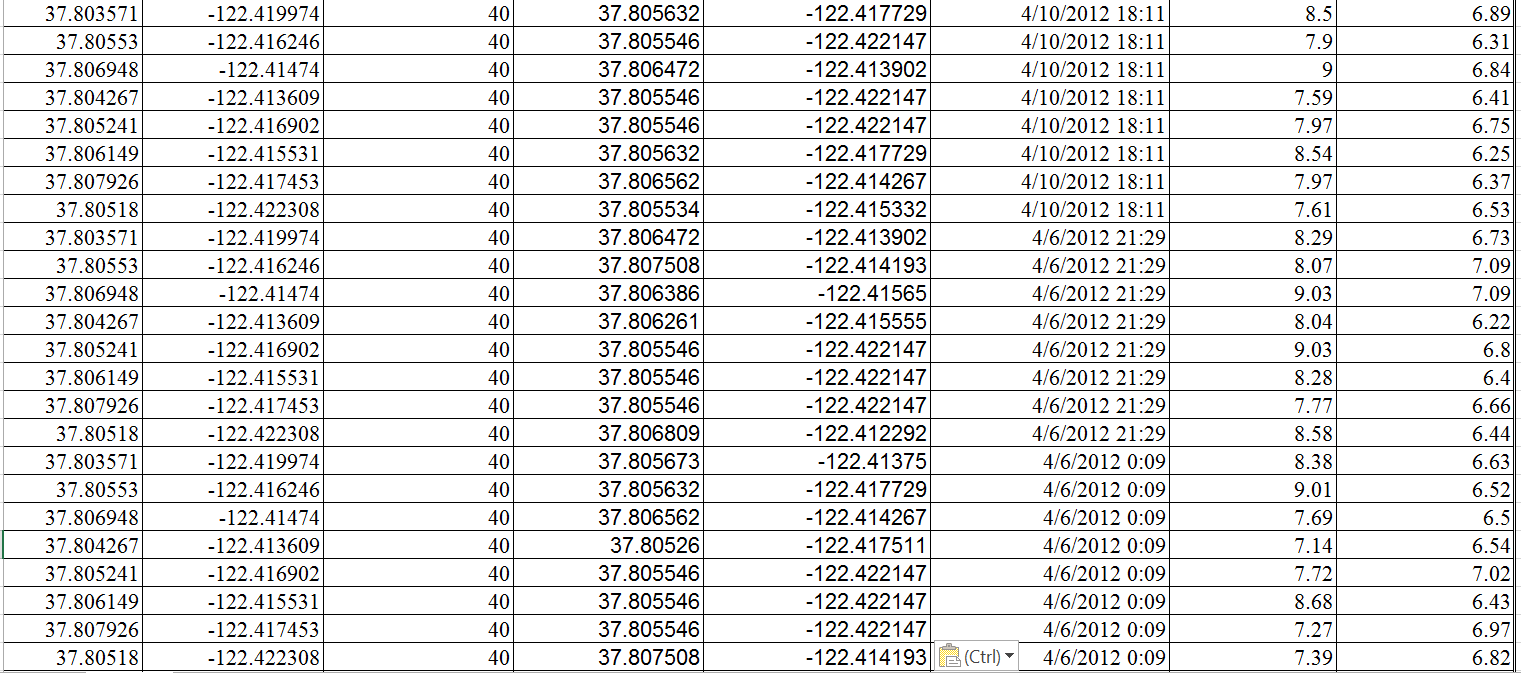
**Figure 23 Test Results for Historic Data Based Algorithm with 20% Congestion**

##### 6.2.5.8.4 Congestion Level – 30%

****

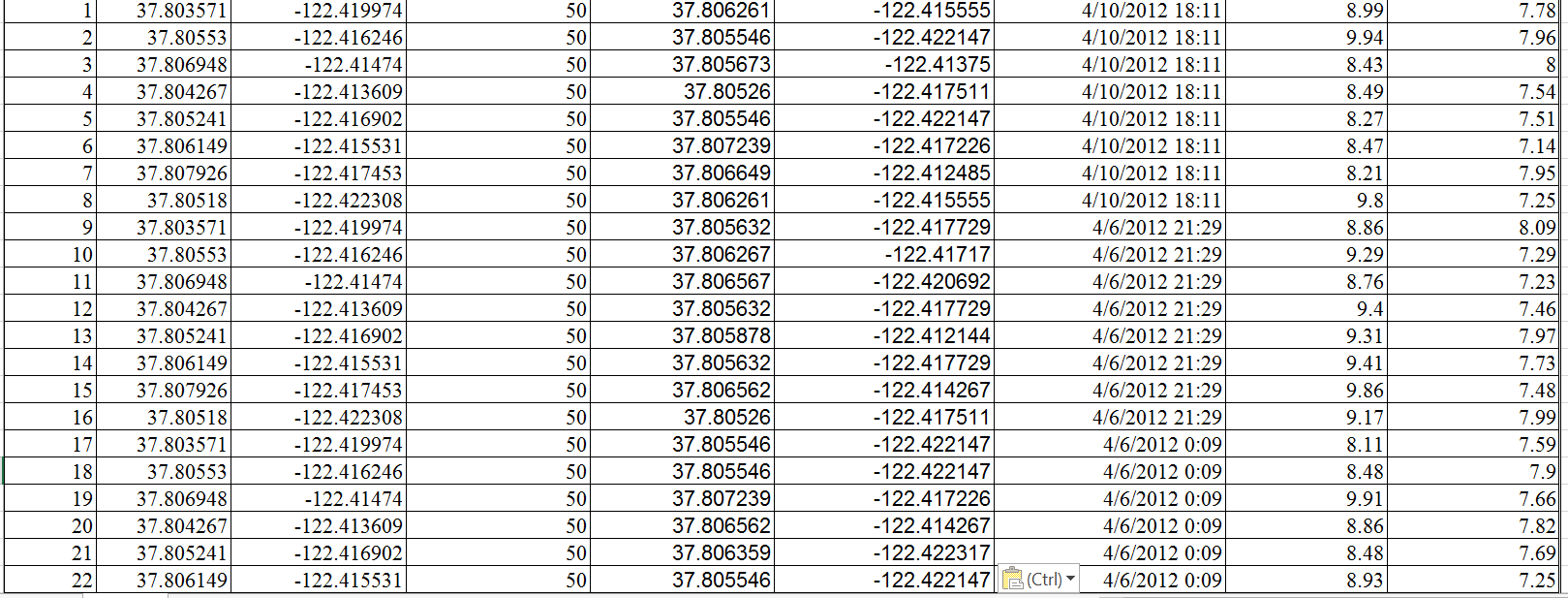
**Figure 24 Test Results for Historic Data Based Algorithm with 30% Congestion**

##### 6.2.5.8.5 Congestion Level – 40%

****

**Figure 25 Test Results for Historic Data Based Algorithm with 40% Congestion**

##### 6.2.5.8.6 Congestion Level – 50%

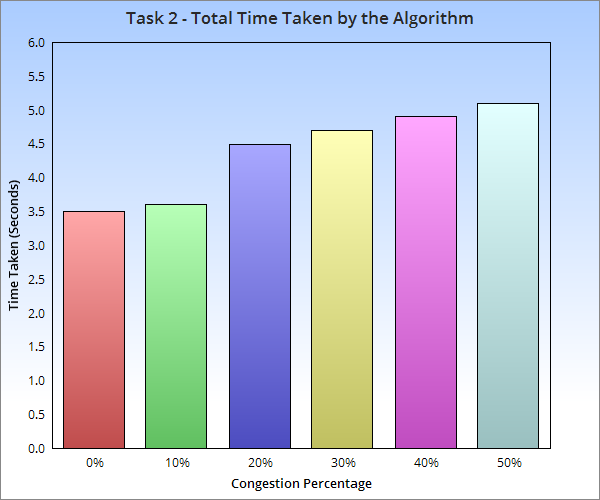


**Figure 26 Test Results for Historic Data Based Algorithm with 50% Congestion**

#### Experiment Analysis

The following Graph is obtained for the above set of test cases

##### Graph1 – Time Taken by the Algorithm to Execute at Every Intersection

****

**Figure 27 Graph between 'Time Taken by Algorithm' and 'Congestion Percentage'**

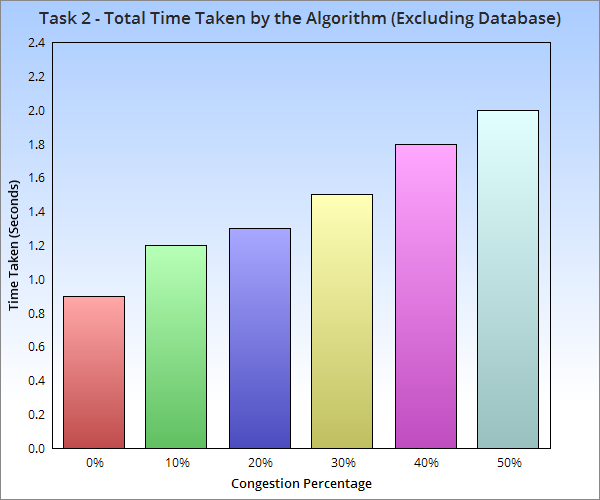
##### Graph Description

The following graph shows the time taken by the algorithm to provide the output to the user (Only Algorithm Time). We have taken the above graph as the sum of the time taken by the algorithm at every intersection. This will give us the execution time of the algorithm under various levels of congestion. For 10% congestion the algorithm takes approximately 2.9 seconds.

|  |
| --- |
| Sum of the Time Taken by the algorithm at every intersection |

This can be considered as the average running time for the algorithm. As the congestion increases the number of available parking slots reduce and thus the time taken by the algorithm to locate the parking slot increases. But as you notice this is not a drastic increase since the algorithm is able to retrieve the next highest weight parking slot as returned by the formula efficiently. Thus the overall time taken by the algorithm here will be determined by the time to retrieve the values from the database.

##### Graph2 – Time Taken by the Algorithm to Execute at Every Intersection (Excluding Database Hit Time)



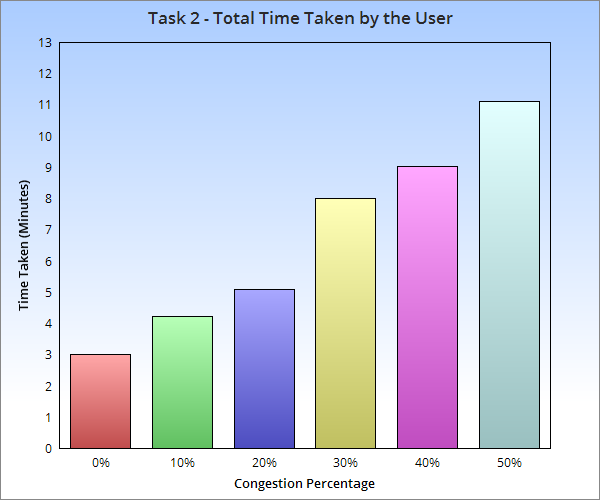
**Figure 28 Graph between 'Total Time Taken by Algorithm' and 'Congestion Percentage'**

##### Graph Description:

In order to make a more accurate description we have also created a graph with the time taken by the algorithm excluding the time taken by the algorithm to hit the database. The above graphs represent the time taken by the algorithm (excluding database hit time).

The time taken by the algorithm is almost constant. This is performed by first measuring the average time taken by the algorithm to retrieve the values from the database .Then the number of times the algorithm hits the database. This is then subtracted from the time taken by the algorithm which then gives us the above graph. This graph is an accurate measurement of the efficiency of the algorithm**.**

##### Graph3 – Total time taken by the user to reach the destination

****

**Figure 29 Graph between 'Time taken to Find Parking' and 'Congestion Percentage'**

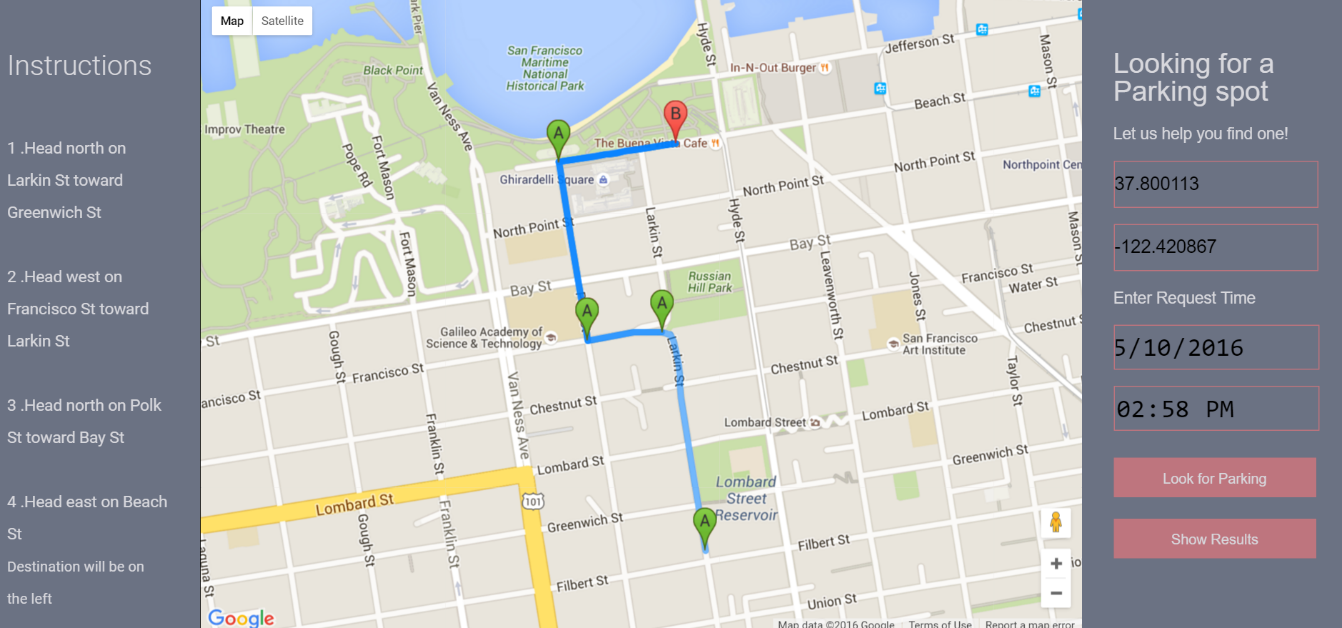
##### Graph Description:

In the graph when the congestion percent is zero the average time taken by the algorithm is 3.2 minutes as a whole which directs the user from the source to the destination. Here time increases logarithmically. Since more the congestion the further the parking slot will be from the user’s location under normal circumstances.

##### Web Output

The following shows the output for the given request time.

The input to this is (37.800113, - 122.420867) and the time it was requested is 05/10/2016 at 02:58 PM



**Figure 30 Output of the Parking Search on the Web application**

## Level of Information - Parking with no data about the parking slots availability.

### 6.3.1. Problem Description

When the users do not have any information or previous experience about the neighborhood, they usually take random paths that are closer to the user’s intended destination. In this task, the real life scenario of Uninformed Search is being replicated. This algorithm will be the baseline algorithm for other algorithms to compare with.

### 6.3.2. Holistic Architecture

This diagram shows the modified version of the system architecture to accommodate the requirements of Algorithm 3

C:\Users\minuf\Downloads\Uninformed_search.png

**Figure 31 Holistic Architecture for Uninformed Search**

### 6.3.3. Alternate Methods Considered

|  |  |  |  |
| --- | --- | --- | --- |
| **No.** | **Method Name** | **Method Description** | **Drawback** |
| 1 | Parking based on nearest intersection, from source, sequentially. | The user is directed to the closest intersection from his original location. Then, if parking is not available in that block, he is directed to the intersection, which is next closest to his original location. This continues till he finds a parking slot, once the blocks are operational. | In this method, the user might be missing on some shortest paths, as he is directed to intersections closest to his original location, every time. |
| 2 | Parking based on nearest intersection, computed incrementally. | The user is directed to the closest intersection from his location, every time, till he finds a parking slot, once the blocks are operational. | In this method, the user may be directed to the slot, far away from his location where he intends to find a parking. |

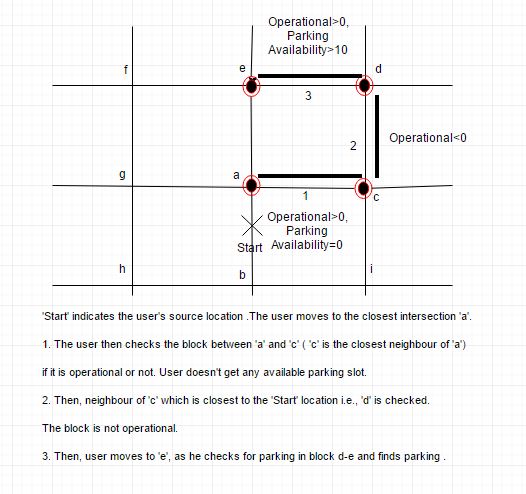
**Table 5 Alternate Methods Considered for Uninformed Search**

### 6.3.4. Method Chosen - Parking Slot Availability estimation based on nearest neighbor to the source location, in a region having pre-defined boundaries.

#### 6.3.4.1. Pre-Cached Data

Data will not be pre-cached since the simulation is based on a user manually looking for a parking slot with no prior information.

#### 6.3.4.2. Algorithm



**Figure 32 Simulation of Uninformed Search Algorithm**

1. The input taken by the algorithm will be the user’s current location and the time at which he wants to find the parking.
2. If the user is not at an intersection, he first moves to closest intersection.
3. A list containing all the intersections visited by the user is maintained.
4. Neighbors of this intersection are computed, and the user moves to any random neighboring intersection. A check is done, if this block is operational. If yes, check for parking slots. The algorithm exits, if the user finds a parking slot here. Else, continue to step 5.
5. Neighbors are computed from the user’s current location (which is an intersection), and he moves to the intersection, closest to his starting point. This process, repeats until the user finds a parking.
6. List of visited intersections is cleared every time the user visits four intersections. This makes sure that the user is not diverted to the block, farther from his starting point (which is the usual real time scenario).

#### Flowchart

Below is the flow chart representing steps for Uninformed Search.

C:\Users\Pooja\Desktop\FlowChart-3 (1).png

**Figure 33 Flow Chart for the Uninformed Search Algorithm**

#### Individual Experiments

The Experiments were conducted based on the following metrics

* Time taken for the algorithm to find a parking spot under various levels of congestion
* Time taken for the user to find a parking spot under various levels of congestion

#### Experiment Inputs

The test data is stored in a file named “TestData.xls”. The csv file contains the following data corresponding to 225 different source – timestamp combinations.

1. Latitude – latitude of the initial position of user

2. Longitude – longitude of the initial position of user

3. Timestamp – Time at which user is searching for parking

#### Experiment Methodology

In this case, experiments are conducted by adding congestion. Congestion is induced by reducing the number of available parking slots by a certain percentage of the maximum possible value of the parking slot availability as given in the database (i.e. 19). The experiment is conducted for various levels of congestion – 10%, 20%, 30%, 40% and 50%. The test data is stored in a file named “TestData.xls”.

The csv file contains the following data:

1. Latitude – latitude of the initial position of user

2. Longitude – longitude of the initial position of user

3. Timestamp – Time at which user is searching for parking

4. Congestion Percent – The congestion percentage that is taken into consideration.

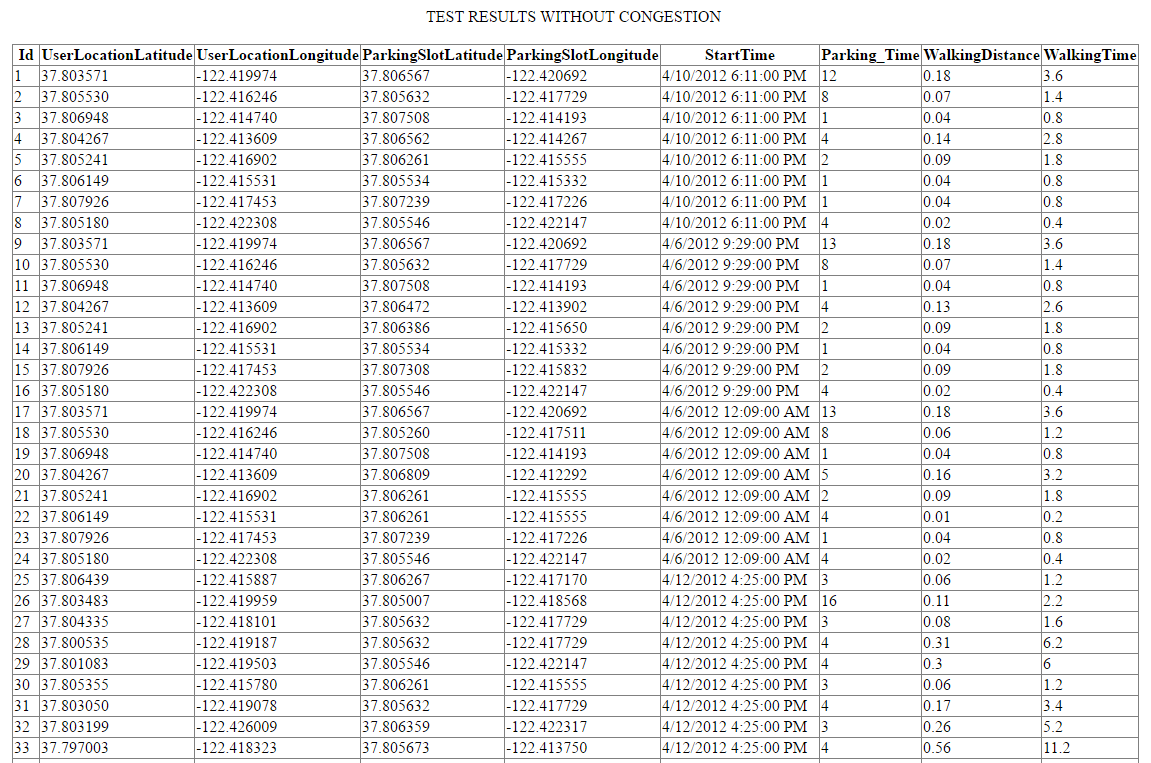
#### Experiment Steps

* In order to find the results of the algorithm, a button named ‘Show Results’ is available. On click of this button, the data in the spreadsheet is taken as input and the algorithm is run by considering the inputs one at a time.
* The results of the test cases are stored in the database and displayed on the UI.

#### Experiment Results

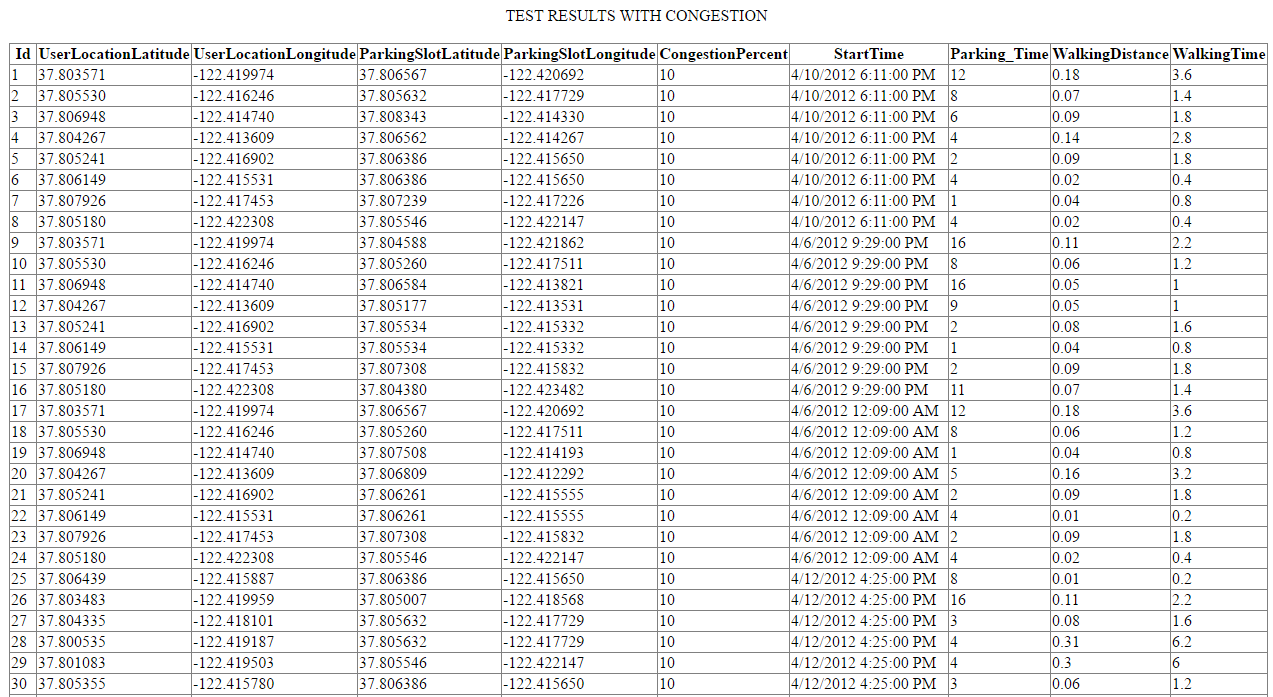
* Results of the experiments conducted for Uninformed Search are displayed below.
* UserLocationLatitude/UserLocationLongitude – Initial start location of User
* ParkingSlotLatitude/ParkingSlotLongitude – Location of the allotted parking block
* StartTime – Time at which user starts searching for a parking
* Parking\_Time(Minutes) – Time taken by the user to find a parking block.
* WalkingDistance(Miles) – Distance taken by the user to walk to destination from the allotted parking block
* Walking Time(Minutes) – Time taken by the user to walk to destination from the allotted parking block

##### 6.3.4.8.1 Congestion Level – 0%



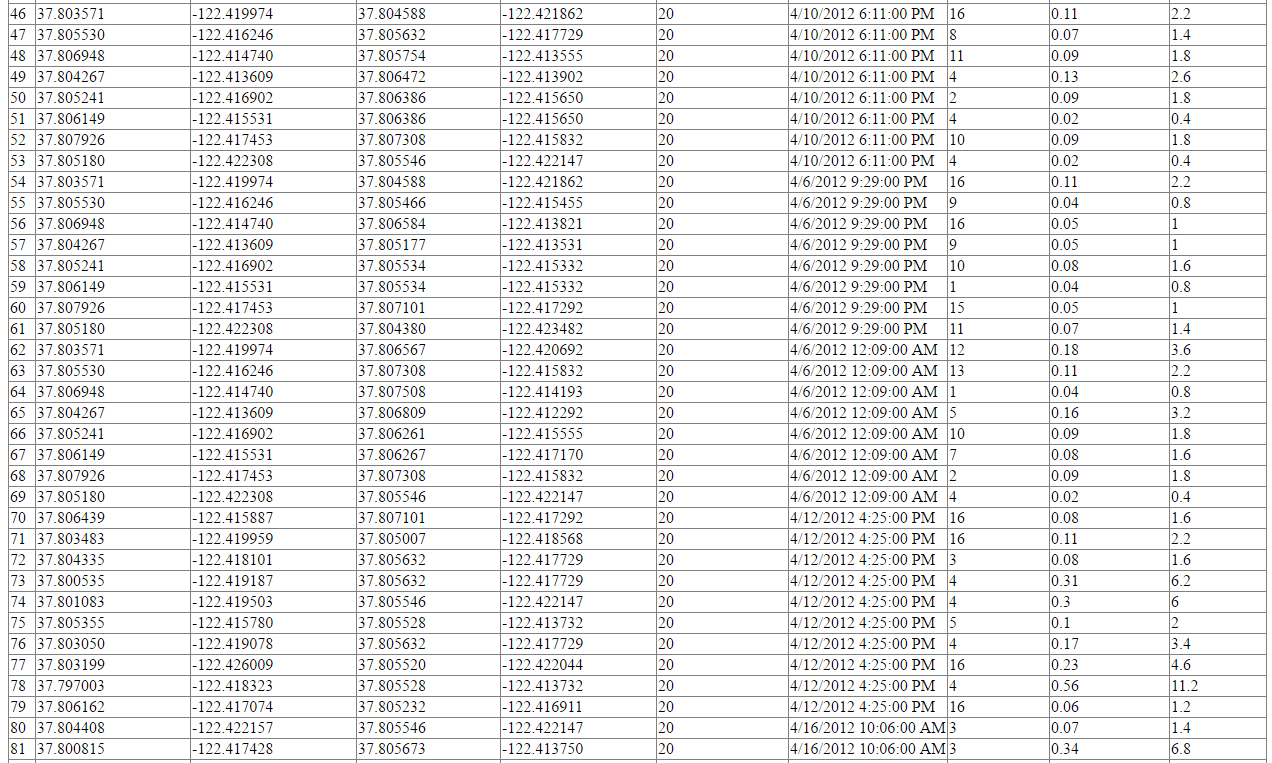
**Figure 34 Test Results for Uninformed Search with 0% Congestion**

##### 6.3.4.8.2 Congestion Level – 10%



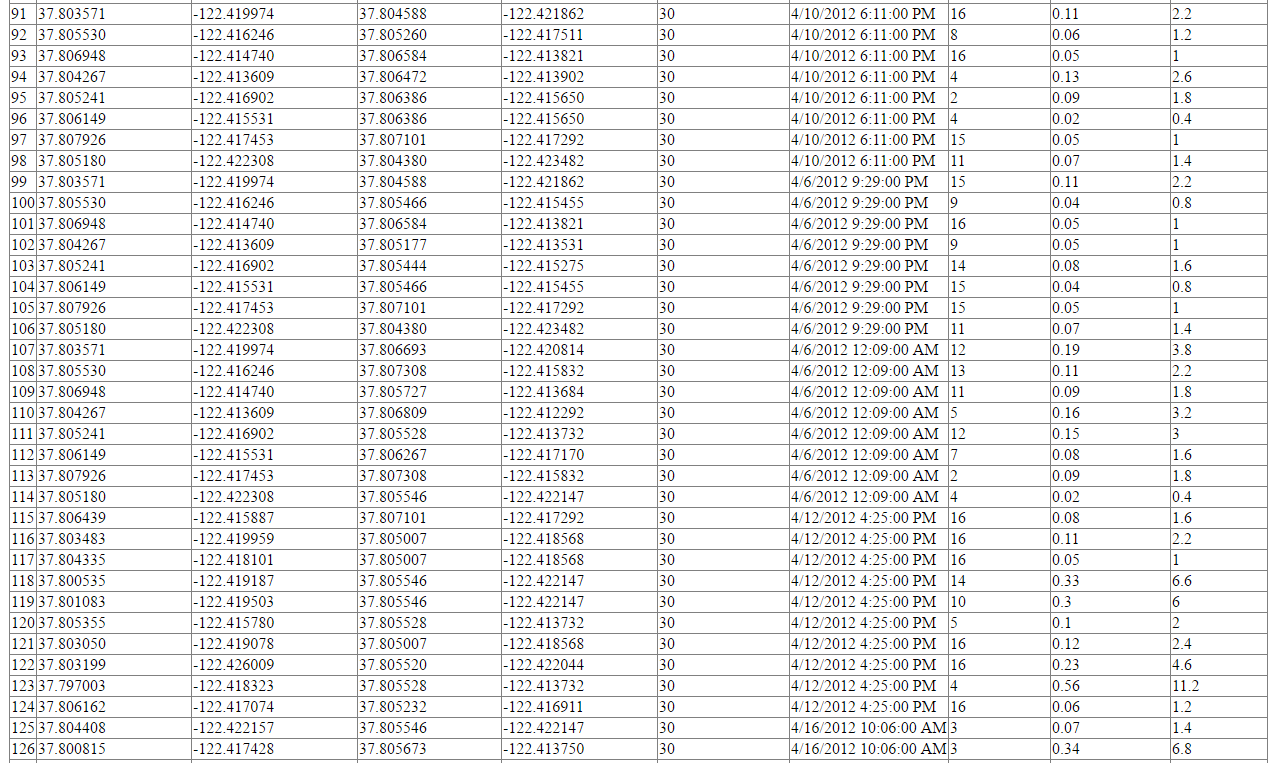
**Figure 35 Test Results for Uninformed Search with 10% Congestion**

##### 6.3.4.8.3 Congestion Level – 20%



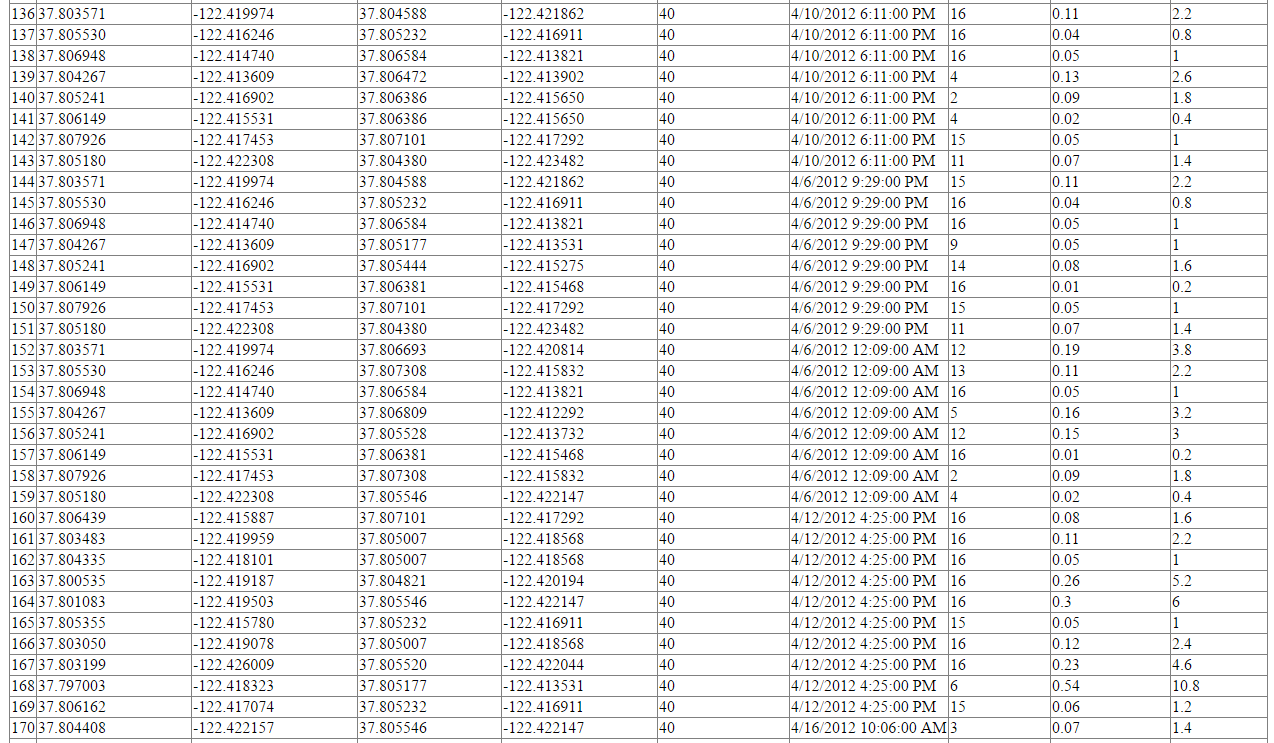
**Figure 36 Test Results for Uninformed Search with 20% Congestion**

##### 6.3.4.8.4 Congestion Level – 30%



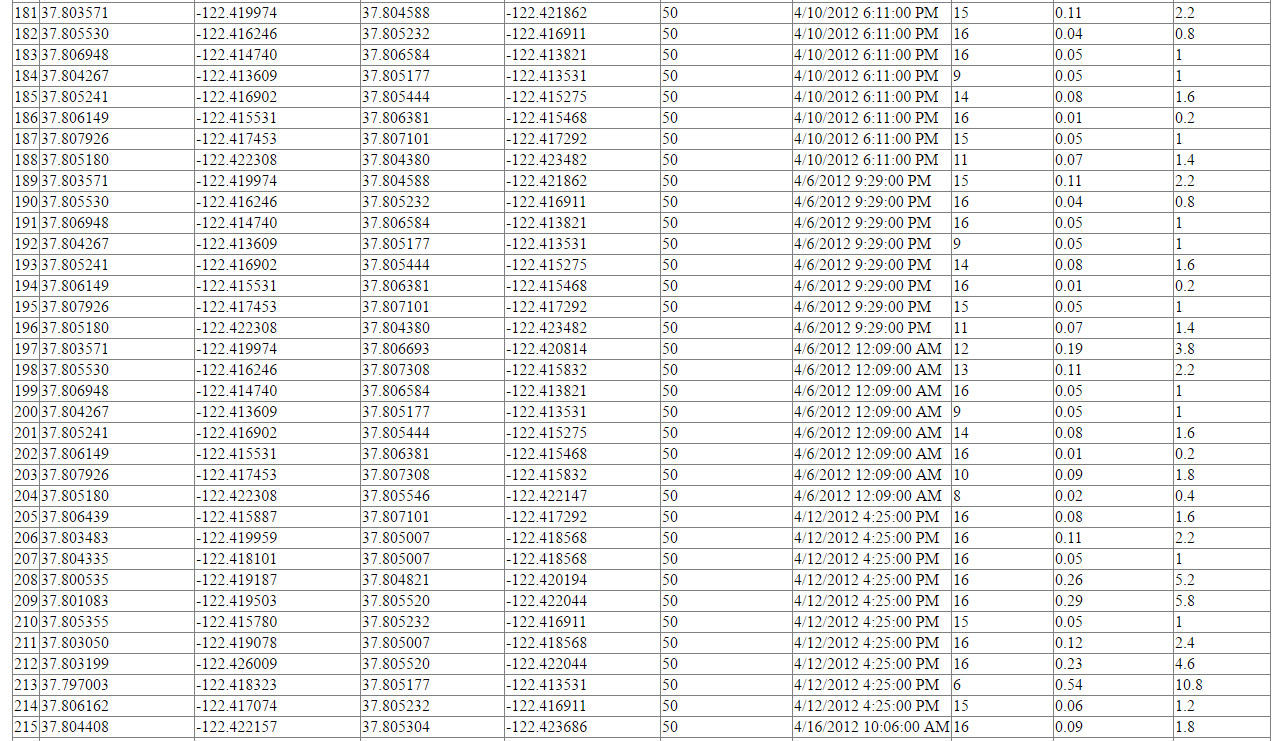
**Figure 37 Test Results for Uninformed Search with 30% Congestion**

##### 6.3.4.8.5 Congestion Level – 40%



**Figure 38 Test Results for Uninformed Search with 40% Congestion**

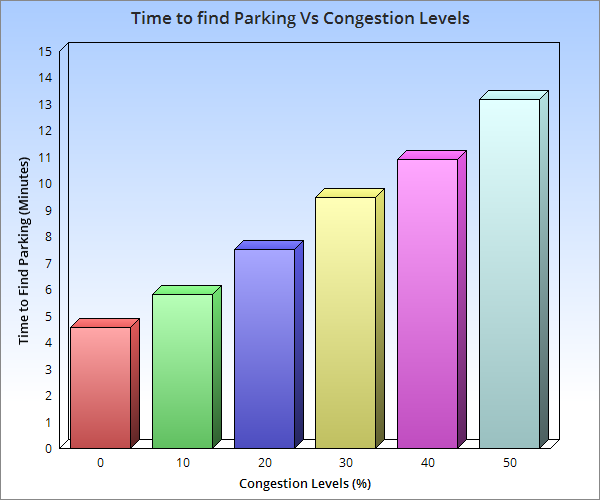
##### 6.3.4.8.6 Congestion Level – 50%



**Figure 39 Test Results for Uninformed Search with 50% Congestion**

#### Experiment Analysis

##### Graph1: Time to find Parking Vs Congestion Levels

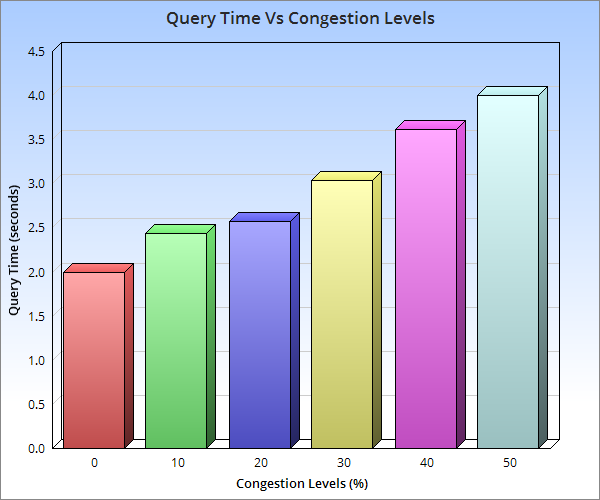
****

**Figure 40 Graph between 'Time to Find Parking' and 'Congestion Levels'**

##### Graph Description

The above graph shows the analysis of ‘Time taken to find parking’ at different ‘Congestion Levels’. From the above graph, we can conclude that the time taken to find a parking slot, on average increases with increase in congestion as the number of available slots decrease.

##### Graph2: Query Time Vs Congestion Levels

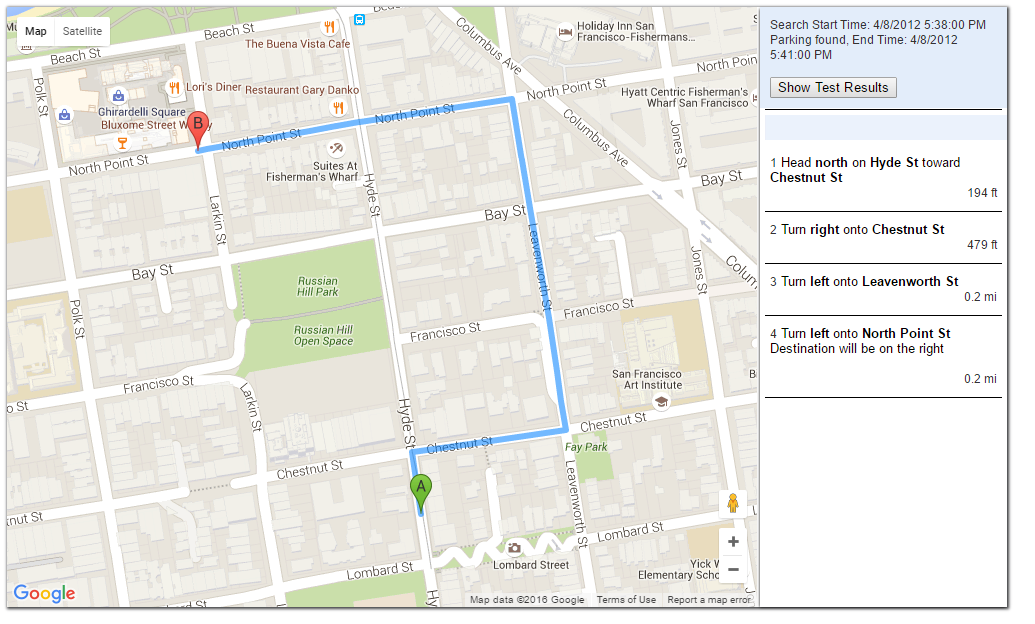
****

**Figure 41 Graph between Query Time and Congestion Levels**

##### Graph Description:

The above graph displays a plot between ‘Query Time’ and ‘Congestion Level’. Query time refers to the running time of the algorithm and this has been recorded for various congestion levels. For each congestion level, the time plotted on the graph is the average of running time of the algorithm over 45 different inputs.

##### Web Output



**Figure 42 Output of the Uninformed Search on Web Application**

## Comparison of algorithms

The above tasks are compared and contrasted with the same set of inputs. The total time taken by the user is noted down for each of the Algorithms. Each of these inputs are mapped against various congestion levels.



**Figure 43 Graph showing the Comparison between various algorithms implemented**

As observed the Real Time Gravitational Algorithm has the best performance as compared to the rest. The Historic Based tries to catch up but fails in some areas since the prediction is based of clubbed historic data which might not be very accurate when there is a wide variety of fluctuation for that parking slot. On the other hand the uninformed search performs quite well when a parking lot is readily available. It lags when the congestion level is increased since the user would have to manually travel and check multiple blocks for available parking slots.

## Technologies Used

These were the technologies used to create the web app

1. ASP.NET
2. C#
3. MYSQL
4. Javascript
5. Python

## Steps to Run the Project

1. Install Visual Studio 2013 Community Version from here

https://www.visualstudio.com/en-us/news/vs2013-community-vs.aspx

1. Download the Zip File from github containing the source code

<https://github.com/francisminu/DBMSFinal.git>

1. Place the ‘Projects' folder present in \DBMSFinal folder on github into the below location:

C:\Users\User\_name\Documents\Visual Studio 2013\ (User\_name : Check the user name of the current machine)

If the folder ‘Projects’ is already present, then copy only ‘DBMSResourceSharingProject’

1. Place the ‘Websites' folder present in \DBMSFinal folder on github into the below location:

C:\Users\User\_name\Documents\Visual Studio 2013\ (User\_name : Check the user name of the current machine)

If the folder ‘Websites’ is already present, then copy only ‘DBMSResourceSharingProject’

1. Open Visual Studio 2013. Open the project by clicking 'Open' and choosing the DBMSResourceSharingProject.sln file present in the location:

C:\Users\User\_name\Documents\Visual Studio 2013\Projects\DBMSResourceSharingProject

(User\_name - Check the user name of the current machine)

1. Install Newtonsoft JSON package via Nugget from the package console manager

(Tools -> Nugget Package Manager -> Package Manager Console)

1. Build the project (Build -> Build Solution)
2. Set index.html as the start page (Right click index.html page in the solution explorer)
3. Click on ‘Run’

## Acknowledgment

We would like to thank professor Wolfson for taking out his time and helping us at every stage of the project development process.

## References

[1]. <https://piazza.com/class_profile/get_resource/ijdvcreqgcn36z/ijupxosubfw24k>

[2]. <https://en.wikipedia.org/wiki/Spatiotemporal_database>

[3]. <https://www.cs.uic.edu/~dayala/mdm12.pdf>

[4]. <http://www.esri.com/news/arcnews/winter0809articles/the-geodatabase.html>

[5]. <http://sfpark.org/how-it-works/open-data-page/>

[6]. <https://developers.google.com/maps/documentation/javascript/>

[7]. <https://developers.google.com/maps/web-services/client-library>