SIMULATION ON SERVICE VEHICLE DISPATCHING

Siddartha Krishnan

A thesis submitted in partial fulfilment of the requirements for the degree of

BACHELOR OF APPLIED SCIENCE

Supervisor: Professor B. Benhabib

Department of Mechanical and Industrial Engineering
University of Toronto

March, 2008
Abstract

Service vehicle dispatching is critical to many industries and emergency services. Improved vehicle dispatch can lead to cost and time savings, and can thus have a significant impact on the quality of the service provided. The focus of this thesis was the development of a simplified computer model and simulation of a typical taxi-dispatching environment. The purpose of this model was to enable testing of various dispatching algorithms to determine their relative performances. The model was written in python and implemented a high degree of modularity so that it can be easily upgraded in terms of complexity, as well as easily adapted to different scenarios. Another objective of this thesis was to develop a basic taxi-dispatching algorithm that implemented vehicle anticipation, which is the process of determining where the next vehicle will likely be needed and pre-emptively satisfying the need. While the implementation of vehicle anticipation was highly simplistic, it did manage to reduce average customer wait times when the number of taxis was fairly small compared to the demand on the system. However with larger numbers of taxis, vehicle anticipation actually increased the average wait time and hence must not be used in these cases. Although the implementation of vehicle anticipation does increase the costs of dispatching, it does show promise for a few specific situations.
Acknowledgements

I would like to thank my thesis advisor, Professor B. Benhabib of the Department of Mechanical and Industrial Engineering, University of Toronto, for his guidance and supervision throughout the course of this thesis, and during preparation of this thesis report.
# Table of Contents

ABSTRACT I  
ACKNOWLEDGEMENTS II  
TABLE OF CONTENTS III  
LIST OF SYMBOLS V  
LIST OF FIGURES VI  
LIST OF TABLES VII  

1 INTRODUCTION 1  
1.1 Motivation 1  
1.2 Background 2  

2 OBJECTIVES AND SCOPE 4  
2.1 Objectives 4  
2.2 Scope 5  

3 LITERATURE REVIEW 6  
3.1 Distance-Based Approaches 6  
3.2 Use of Computer Based Agents 7  
3.3 Use of Heuristics 8  
  3.3.1 The Tabu Search Algorithm 8  
  3.3.2 Other Search Heuristics 9  
3.4 Markov Models 10  
3.5 Summary 11  

4 METHODOLOGY 12  
4.1 Model development 12  
  4.1.1 Overview of the Model 12  
  4.1.2 Assumptions 14  
  4.1.3 Variables 15
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.1.4 Service Objectives</td>
<td>16</td>
</tr>
<tr>
<td>4.1.5 Structure and Programming Details of the Model</td>
<td>17</td>
</tr>
<tr>
<td><strong>4.2 Dispatch Algorithm Development</strong></td>
<td>27</td>
</tr>
<tr>
<td>4.2.1 General Approach</td>
<td>27</td>
</tr>
<tr>
<td>4.2.2 Baseline Algorithm - Random Dispatch</td>
<td>29</td>
</tr>
<tr>
<td>4.2.3 Vehicle Anticipation Dispatch</td>
<td>29</td>
</tr>
<tr>
<td><strong>4.3 Algorithm Simulation and Evaluation</strong></td>
<td>31</td>
</tr>
<tr>
<td>4.3.1 Simulation of inputs</td>
<td>31</td>
</tr>
<tr>
<td>4.3.2 Simulation Outputs</td>
<td>32</td>
</tr>
<tr>
<td>4.3.3 Variation of Inputs and Simulation Parameters</td>
<td>32</td>
</tr>
<tr>
<td>4.3.4 Simulation Specifics</td>
<td>33</td>
</tr>
<tr>
<td><strong>5 SIMULATION RESULTS AND ANALYSIS</strong></td>
<td>34</td>
</tr>
<tr>
<td>5.1 Data Processing</td>
<td>34</td>
</tr>
<tr>
<td>5.2 Average Customer Waiting Time</td>
<td>35</td>
</tr>
<tr>
<td>5.2.1 Base Case</td>
<td>35</td>
</tr>
<tr>
<td>5.2.2 Effect of Increase in Frequency</td>
<td>36</td>
</tr>
<tr>
<td>5.2.3 Effect of Increase in Grid Size</td>
<td>37</td>
</tr>
<tr>
<td>5.2.4 Further Analysis of Average Waiting Time Results</td>
<td>37</td>
</tr>
<tr>
<td>5.3 Average Chartered Distance Ratio</td>
<td>38</td>
</tr>
<tr>
<td>5.3.1 General Trends</td>
<td>39</td>
</tr>
<tr>
<td>5.3.2 Effect of Increasing Frequency of Calls and Grid Size</td>
<td>40</td>
</tr>
<tr>
<td>5.4 Further Analysis</td>
<td>41</td>
</tr>
<tr>
<td><strong>6 CONCLUSION AND FUTURE DEVELOPMENT</strong></td>
<td>43</td>
</tr>
<tr>
<td>6.1 Development of the Taxi-Dispatching Environment</td>
<td>43</td>
</tr>
<tr>
<td>6.2 Vehicle Anticipation Algorithm</td>
<td>44</td>
</tr>
<tr>
<td><strong>7 REFERENCES</strong></td>
<td>47</td>
</tr>
<tr>
<td><strong>FIGURES AND TABLES</strong></td>
<td>49</td>
</tr>
<tr>
<td><strong>APPENDIX A: SOURCE CODE</strong></td>
<td>69</td>
</tr>
</tbody>
</table>
List of Symbols

num_taxi: Number of taxi's used in the simulation

grid_size_x: Length of grid (in km)

grid_size_y: Width of grid (in km)

dead_time: Length of time the simulation must run for (in hours)

call_prob: Probability that an incoming call will occur at any given time interval

start_time: Starting time of the simulation

max_time_step: The maximum length of a time interval

time_intvl: The length of a given time interval

incoming_rush_start: Start Time for Rush hour going into Downtown

incoming_rush_end: End Time for Rush hour going into Downtown

outgoing_rush_start: Start Time for Rush hour going out of Downtown

outgoing_rush_end: End Time for Rush hour going out of Downtown

rush_call_increase_factor: The number of times call frequency will increase during rush hour

rush_call_prob: The probability that any given call will be going in the direction of the rush hour traffic

taxi_speed: Speed of taxis (in km/hour)
List of Figures

FIGURE 1 - AVERAGE CUSTOMER WAITING TIME VERSUS NUMBER OF Taxis for Base Case 49

FIGURE 2 - AVERAGE CUSTOMER WAITING TIME VERSUS NUMBER OF Taxis for Double Call Frequency Case 50

FIGURE 3 - AVERAGE CUSTOMER WAITING TIME VERSUS NUMBER OF Taxis for Double Grid Size Case 51

FIGURE 4 - CHARTERED DISTANCE RATIO VERSUS NUMBER OF Taxis for the Base Case 52

FIGURE 5 - CHARTERED DISTANCE RATIO VERSUS NUMBER OF Taxis for the Double Call Frequency Case 53

FIGURE 6 - CHARTERED DISTANCE RATIO VERSUS NUMBER OF Taxis for the Double Grid Size Case 54

FIGURE 7 - FLOWCHART OF SIMULATION STEPS 55
List of Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>TABLE 1</td>
<td>AVERAGE CUSTOMER WAITING TIME (HOURS) USING RANDOM DISPATCH FOR BASE CASE</td>
<td>56</td>
</tr>
<tr>
<td>TABLE 2</td>
<td>AVERAGE CUSTOMER WAITING TIME (HOURS) USING VEHICLE ANTICIPATION FOR BASE CASE</td>
<td>57</td>
</tr>
<tr>
<td>TABLE 3</td>
<td>AVERAGE CUSTOMER WAITING TIME (HOURS) USING RANDOM DISPATCH FOR DOUBLE CALL FREQUENCY CASE</td>
<td>58</td>
</tr>
<tr>
<td>TABLE 4</td>
<td>AVERAGE CUSTOMER WAITING TIME (HOURS) USING VEHICLE ANTICIPATION FOR DOUBLE CALL FREQUENCY CASE</td>
<td>59</td>
</tr>
<tr>
<td>TABLE 5</td>
<td>AVERAGE CUSTOMER WAITING TIME (HOURS) USING RANDOM DISPATCH FOR DOUBLE GRID SIZE CASE</td>
<td>60</td>
</tr>
<tr>
<td>TABLE 6</td>
<td>AVERAGE CUSTOMER WAITING TIME (HOURS) USING VEHICLE ANTICIPATION FOR DOUBLE GRID SIZE CASE</td>
<td>61</td>
</tr>
<tr>
<td>TABLE 7</td>
<td>AVERAGE CHARTERED DISTANCE RATIO USING RANDOM DISPATCH FOR BASE CASE</td>
<td>62</td>
</tr>
<tr>
<td>TABLE 8</td>
<td>AVERAGE CHARTERED DISTANCE RATIO USING VEHICLE ANTICIPATION FOR BASE CASE</td>
<td>63</td>
</tr>
<tr>
<td>TABLE 9</td>
<td>AVERAGE CHARTERED DISTANCE RATIO USING RANDOM DISPATCH FOR DOUBLE FREQUENCY CASE</td>
<td>64</td>
</tr>
<tr>
<td>TABLE 10</td>
<td>AVERAGE CHARTERED DISTANCE RATIO USING VEHICLE ANTICIPATION FOR DOUBLE FREQUENCY CASE</td>
<td>65</td>
</tr>
<tr>
<td>TABLE 11</td>
<td>AVERAGE CHARTERED DISTANCE RATIO USING RANDOM DISPATCH FOR DOUBLE GRID SIZE CASE</td>
<td>66</td>
</tr>
<tr>
<td>TABLE 12</td>
<td>AVERAGE CHARTERED DISTANCE RATIO USING VEHICLE ANTICIPATION FOR DOUBLE GRID SIZE CASE</td>
<td>67</td>
</tr>
<tr>
<td>TABLE 13</td>
<td>SIMULATION PARAMETERS FOR BASE CASE, DOUBLE FREQUENCY CASE, AND DOUBLE GRID SIZE CASE</td>
<td>68</td>
</tr>
</tbody>
</table>
1 Introduction

1.1 Motivation

Several industries and services depend on service vehicle dispatching – ambulances, police, delivery services, and even airlines to name a few. When considering emergency services, efficient dispatching of service vehicles could save lives. For businesses, there are huge cost savings if vehicles are dispatched more efficiently. These savings are mainly due to reduced fuel consumption and time savings. For example: to route and dispatch its aircraft, the courier service UPS uses an algorithm called VOLCANO (Volume, Location, Aircraft Network Optimiser), which the company estimates has saved it tens of millions of dollars since its implementation [i]. Efficient dispatching also helps boost the reliability of the service, which could lead to an increase in market share in a consumer-oriented service. The end result is increased profit for the company and a more reliable service for the customer.

Most current systems of vehicle dispatch rely on human operators to decide how vehicles are dispatched. They depend on the human operator’s experience with little or no guidance from a computer. This has numerous drawbacks, namely the human operator cannot take into account many factors and perform quick calculations needed to make the process most efficient. Human operators are at risk of making biased decisions and can let their personal conditions (such as tiredness) affect their judgement. Thus automated dispatch, where a computer is used to make rational and objective decisions, has definite advantages over a human operator. A computer can take more variables into account when considering a dispatch decision in a complex scenario. Furthermore, human operators require training and hands-on experience before being able to perform the
dispatch task properly, which is both time-consuming and expensive. A computer algorithm can be implemented in a much shorter time period and with immediate results.

This thesis will concentrate on automated dispatch of taxis. In Canada alone, the taxi industry exceeds $1.3 billion dollars [ii]. Increasing the efficiency of taxi dispatching will result in huge savings and increased profits, not to mention a more reliable taxi service. The increased dispatch efficiency will also likely reduce the resources required to run the taxi network and hence decrease the burden on the environment (due to reduced fuel consumption and less emissions from taxis). The taxi dispatch problem is more complicated than many other service vehicle dispatch problems as there is no central place where a taxi will be directed to after responding to a request for service (such as a hospital in the case of ambulance dispatch). A taxi can be called for a pickup anywhere, to go to anywhere. This adds another dimension to the problem. This added complexity allows the solutions derived for taxi dispatch to be applied to less complicated service vehicle dispatch problems.

1.2 Background

Taxi networks generally consist of a fleet of independently moving taxis communicating with a centralised control centre. The control centre accepts requests for service from consumers through various mediums (phone, web-based, etc). The control centre then relays dispatch information verbally to taxis using either radio or GPRS communication.

Currently, most companies rely on manual dispatching systems or computer-assisted dispatch to dispatch their taxis. Manual dispatching relies solely on the human operator’s experience to make the dispatch decision. Computer-assisted dispatching uses
real-time vehicle tracking (using Global Positioning System or GPS) to display information on a geographical map. It may also perform rudimentary distance calculations. The operator still needs to make the final dispatch decision. Several commercial products, such as the *TaxiMan* system used by Beck Taxi [iii], exist in the market. However, there is a definite dearth of fully automated dispatch software.

Tracking the locations of vehicles in real-time, using GPS, is now widespread in many vehicle fleets. This combined with the availability of computer terminals and GPRS communications in taxis would make implementation of automated dispatch systems reasonably easy. Assignment of taxis to requests for service could be done using an automated dispatch algorithm run at the control centre and the dispatch directions relayed electronically via GPRS to the individual taxi terminals.
2 Objectives and Scope

2.1 Objectives

The primary objectives of this thesis are:

- To create a simplified computer model and simulation of a typical taxi dispatching environment (the city where the dispatching occurs, the incoming calls for service, etc.) and taxi dispatching system (how the taxis are allocated and dispatched). This simulation must be able to run multiple dispatching algorithms and serve as a platform to evaluate these multiple dispatching algorithms with respect to a set of service objectives.

- To conduct a literature review to learn more about service vehicle dispatching, paying special attention to different algorithms for automated dispatch.

- To develop a simple algorithm to dispatch service vehicles throughout a given area using vehicle anticipation. Vehicle anticipation refers to predicting where the next vehicle will likely be needed, based on a known probability distribution of calls.

- To simulate the vehicle anticipation algorithm developed within the dispatching environment model framework to determine the performance of our automated dispatch algorithm. The performance of the dispatching algorithm will be evaluated on a set of service objectives (e.g. to minimise average waiting time for a call). The algorithm developed will be compared to a baseline algorithm, where taxis remain where they are till they are directed to a call and the closest available taxi to a call will be assigned to the call.
2.2 Scope

As mentioned previously, this thesis will focus on taxi dispatching. The dispatching environment for taxis, that is the city in which the taxis are being dispatched and the external factors influencing taxi dispatch, are extremely complicated. The dispatching environment model developed in this thesis will attempt to simulate some of the basic features of a typical dispatching environment under ideal conditions. The features taken into consideration will try and encompass the most pertinent elements of a typical taxi dispatch scenario, in order to determine the major resource constraints for the taxi dispatching system to handle. The inputs to the system are simplified, both in terms of how they are generated, as well as how they can be satisfied. Finally, the dispatching system itself is simplified in terms of the information it handles, as well as how dispatch decisions are made. The service objectives against which dispatching algorithms will be evaluated are simpler and fewer in number than in a typical dispatching system at a taxi company. However, it is important to note that the functions performed by each of these elements of the simulation is similar to the function it would perform in a realistic dispatching scenario, but with fewer complexities taken into consideration. The purpose of the simulation is not to evaluate specifics of an automated dispatch system, but rather to determine general relative performance of different algorithms. A full list of assumptions on the dispatching environment and the dispatching system is presented in Chapter 4: Methodology of this thesis report, under Section 4.1.2: Assumptions.
3 Literature Review

An in-depth literature review was conducted to assess current solutions and alternate algorithms to the vehicle dispatch problem. This literature review helped determine how to approach the design of an automated dispatch algorithm and what factors to take into account when creating the model of the dispatching environment. The literature review was also a starting point to determine how to implement vehicle anticipation (for example, whether it will implement it by anticipating calls based on past data or by dividing the city into several sectors and making sure no sector is unoccupied).

3.1 Distance-Based Approaches

The starting point of the literature review was a thesis on the topic of service vehicle dispatching completed in 2005 with the same supervisor, Professor B. Benhabib [iv]. This thesis compares the use of a nearest-taxi dispatching method, where the requests for service are answered chronologically (in the order they come in) by assigning the nearest (in terms of distance to service request location) available taxi to them, to the use of a random vehicle selection algorithm, where a random available taxi is assigned to answer the request for service (without taking into consideration distance of the vehicle from the origin of the request for service). As was hypothesised, the nearest-taxi dispatching method resulted in a shorter average distance travelled by the taxis to the request for service location and hence a quicker average response time. This simulation did not take into account several factors, such as differences in the travelling times between different regions (e.g. travelling in the downtown region might be slower than in the suburbs), or differences in demand patterns over geographic region or time (requests
for service were generated in random locations at random times). Furthermore, the algorithm developed did not take into account different service objectives, but rather only satisfied the objective of minimum customer waiting time. This thesis is however a good starting point to approach the problem and can be built upon by incorporating more complexities and a more sophisticated (and perhaps more efficient) algorithm.

Koster et al [v] discuss the routing and dispatching of guide vehicles in a factory using distance based rules. Their findings indicate that assigning the closest available vehicle to a request for service results in the lowest average waiting time before the request is satisfied, but could have a relatively high maximal waiting time. They determine that time-truncation, in which priority is given to loads that wait more than a certain time threshold, helps to lower the maximal waiting time without much increase in average waiting time. This would be a good method to implement and see if it increases the efficiency of taxi dispatch. It was also found that a first-come-first-served method of allocating guide vehicles, where calls were answered in chronological order by the first available vehicle performed the worst. While this paper considers routing and dispatch of guide-vehicles in a factory, the basic problem is the same as that of taxi dispatch and hence the findings should be applicable to taxi dispatch.

### 3.2 Use of Computer Based Agents

Benyahia and Potvin [vi] discuss a framework for vehicle dispatch. Their framework consists of several complex agents organised in a hierarchical fashion. Each agent concentrates on a specific function of dispatch (e.g. monitoring the environment and traffic) and information passes upwards in the hierarchical chain. According to their findings, this method of information management is suitable for dynamic dispatching. It
also allows information to be compartmentalised and hence allows for modularity and
easy extension/expansion. Modularity is important as separate modules could speed up
the development process and allow individual modules to be upgraded or replaced
without affecting the rest of the program.

Seow et al [vii] discuss the idea of using individual agents in a taxi-dispatching
scenario. Their implementation involves grouping requests for service by geographical
region and by specific time windows. The service requests are then assigned to vehicles
in the same geographic region. The vehicles are independent agents, which can
collaborate with each other, and have the power to change dispatch assignments amongst
themselves. These agents negotiate on behalf of each taxi and switch dispatch
assignments such that the groups average satisfaction is maximised (i.e. waiting time is
minimised). Their findings show that regardless of group size, waiting time and empty
cruising time decrease.

3.3 Use of Heuristics

Even when the dispatch problem can be simplified enough that an exact solution is
possible, it is likely to take too long to compute on most hardware. Hence the use of
heuristics could yield a good solution with less computational effort.

3.3.1 The Tabu Search Algorithm

Cordeau and Laporte [viii] have conducted a literature survey on different
heuristics utilised in solving the vehicle dispatch and vehicle routing problems, and have
indicated that the Tabu search algorithm stands out as a promising candidate. The Tabu
search algorithm uses the localised search algorithm to parse through solutions to the
dispatch problem till the solution criteria are met. Once a potential solution is found, it is stored in a memory structure and all future searches omit solutions which are similar to those stored in the memory structure (and hence reduce the possibility of searching for similar solutions). Thus the solution space is traversed in less time and a reasonably optimal solution is found (rather than looking through all possible solutions which is time consuming). Gendrea et al [ix] apply the Tabu search algorithm to delivery dispatching with good results. Their dispatching scenario has multiple pickup and delivery locations, which are generated in real-time, thus their scenario is similar to that of a typical taxi-dispatching scenario. They also state that the Tabu search heuristic can cope with complex and dynamic environments present in dispatching scenarios, such as taxi dispatching.

3.3.2 Other Search Heuristics

Apart from Tabu search, several other heuristics have been implemented to try and optimise the vehicle dispatch problem. Gillett and Miller [x] discuss a sweep algorithm for large-scale vehicle dispatch. Previous solutions were limited to 100 locations (service locations) since they treated the dispatch scenario as one big problem. Their solution divides the solution space into several routes. Locations are arbitrarily assigned to routes and then locations on routes are replaced with locations from other routes if total distance travelled decreases. This continues till no improvement is possible and then goes on to the next route. This continues till all routes are optimised. The process is then repeated backwards and the best solution (forwards or backwards) is the one with the minimum total distance travelled.
Weintraub et al [xi] use a priority-based heuristic to improve the dispatch efficiency of service vehicles for an electric grid. The priority of any given request for service is based on the time elapsed till a service vehicle arrives on the site. The distances travelled are weighted depending on the priority of other calls. The total distance travelled (weighted on priority) is minimised. The geographic area is first broken down into several regions and the requests for service in the areas are divided into several routes. Three nodes are assigned arbitrarily to each route. Then an additional node of high weighting is added in a position that minimises travel time. This is repeated till all nodes are added to the route. This method is good as it produces routes quickly and can easily incorporate new nodes. Nodes are based on current requests for service as well as requests for service based on past data.

3.4 Markov Models

Markov models are another way of representing the dispatch scenario. The requests for service and the pickups off the street are occurrences that happen with a fixed probability and do not depend on previous states of the system (i.e. whether there was a request for service previously). Hence these can be modelled as Markov processes. Larson and McKnew [xii] used Markov models to model the events of a police patrol dispatch scenario. The patrol-initiated activities and requests for service are both modelled as independent Poisson processes. Minkoff [xiii] also successfully uses a Markov model to model a mid-sized delivery service dispatch system. Rather than obtaining an exact solution to the problem, Minkoff uses a heuristic to solve it.
3.5 Summary

The literature review provided a good starting point for the development of the automated dispatch algorithm as information on algorithms was plentiful. However, information on the critical factors and parameters of the taxi-dispatching environment was hard to come by and hence many of these values used in our simulation were estimated. Information on vehicle anticipation was difficult to come by as well, however, the topic was touched upon in some sources that discussed vehicle-patrolling strategies. While much of the information researched could not be implemented into the final vehicle anticipation algorithm, it does serve to increase knowledge of vehicle dispatching algorithms and give a general direction to the development of the algorithm.
4 Methodology

This section outlines the approach taken to achieve the thesis objectives. First, the dispatching environment and the dispatching system was modelled. The dispatching system interfaces with the dispatching environment model and allows different dispatching algorithms to be used. Then different dispatching algorithms were developed to allocate how taxis would be allocated. Finally a simulation was performed, to compare performance of the dispatching algorithms and to determine the effects changes in inputs will have on changes in performance.

4.1 Model development

The model of the dispatching environment and dispatch system is one of the key deliverables of this thesis. The model will be used to test different automated dispatch algorithms developed and evaluate their relative performance in satisfying the service objectives.

4.1.1 Overview of the Model

The model will consist of several modules, each taking on a specific function/task. The different modules will interface with the main simulation module, which then passes information between modules and runs the simulation.

The main simulation module controls the time of the simulation. It generates time intervals of semi-random lengths to simulate calls arriving at random times. At the start of each interval, calls are generated by the call-generation module and passed to the main simulation module. Similarly, the taxi module passes the locations and statuses of all taxis to the main simulation module. The call information and taxi locations/statuses are
then passed by the main simulation module to the dispatch module. The dispatch module then makes decisions on how the taxis should be assigned to answer each call, as well as how idle taxis should behave, and passes these back to the simulation module. The simulation module then passes on these assignments to the taxi module, which updates the destinations and statuses of the individual taxis. The taxi module then moves each taxi till the time interval is completed, or till a taxi has reached its destination. In the case that the taxi has reached its destination before the time interval is over, the taxi module passes this information on the to simulation module, which in turn passes this on the to dispatch module. The dispatch module then decides how the taxi should behave (e.g. should it travel idle somewhere, or go pickup another call), and relays it back to the taxi module. If the interval is over, a new time interval is created and the process repeats, till the end time of the simulation is reached. A chronological set of steps is presented in Section 4.1.5.3.1 of this chapter. A flow chart of the entire process is provided in Figure 7 – Flowchart of Simulation Steps in the Figures and Tables chapter of this report.

The following key points pertaining to the model must be noted before each module can be discussed in more detail:

- The city (area to be dispatched in) will consist of a rectangular grid. The city will be divided into several geographic zones, which relate to different regions of a city (e.g. downtown region, suburbs, etc.).
- Taxis will be allowed to travel anywhere on the grid, within the grid boundaries. Taxis will not have to follow any given path (i.e. there are no “roads” in the area for the taxis to follow – they can take a straight path to their target).
• Idle taxis can be only occupied when they answer a request for service (a call). They cannot be occupied by a pickup from the “street” (i.e. they cannot be flagged down by customers while they are travelling idle). A customer can call and request a taxi to pick them up at a given location and drop them off at any another location on the map.

• The simulation will have a timed element. Demand (in-terms of incoming requests for service) will vary according to both geographic zones and time of day (e.g. higher demand at rush hour).

• The model will have two rush hours:
  o An incoming rush hour where the majority of calls will be originating from the suburbs of the city and will want to end in the downtown sector
  o An one outgoing rush hour where the majority of the calls will originate in the downtown sector and will want to travel to the suburb sectors.

The model was developed such that modules are self-contained. Modularity will enable easy expansion and more versatility in the model. Separate modules must be able to work independently of each other such that they can be replaced/upgraded as needed. It is particularly important that the dispatch module itself can be changed, to allow several different dispatch algorithms to be compared. Modularity also allows modules to be tested individually to ensure there are no programming errors and to simplify debugging if errors occur.

4.1.2 Assumptions

As mentioned previously, several assumptions were made to reduce the complexity of the model. The key assumptions are listed below:

• The city is rectangular in shape and completely two-dimensional
• The taxis can move anywhere in the grid (i.e. they can travel in a straight line to their destination, without any obstructions)

• Taxis can only be chartered when they answer a request-for-service call

• Discrete time intervals are used

• Velocity of vehicles throughout the simulation is constant.

• Vehicle positions and status are known at all times (this is a realistic assumption considering GPS tracking capabilities which are prevalent in most fleets of service vehicles)

• Traffic obstructions are not taken into consideration in this model (e.g. traffic lights, accidents, etc)

• A vehicle is ready for new job as soon as it becomes idle. There is no time delay when a vehicle picks up or drops off a customer

• Once a request for service is assigned to a given vehicle it will not be changed mid-way (this is a reasonably sound assumption, as the taxi driver is unlikely to want to give up his assigned service request once he has started to travel towards it).

• Initial vehicle locations are assigned randomly

4.1.3 Variables

There are several parameters of the model, which can be manipulated as desired.

The major ones are listed below:

• Number of taxis in the fleet

• Size of grid (city)

• Types, demand probabilities, locations, and sizes of the geographic regions (sectors) in the city
- Speed of the vehicles
- Probability that there will be an incoming call (request-for-service) at any given time interval
- The maximum length of time of the time intervals
- The length of time the simulation can be run for
- The timings of rush hours and their proportion of traffic that is moving in the rush hour direction

These parameters can be changed to change the scale of the simulation and adapt it to specific scenarios (e.g. specific cities, specific taxi fleets).

### 4.1.4 Service Objectives

Service objectives are the goals that the dispatch algorithm is trying to achieve. These would be set by the company that owns the fleet of taxis. How well the different algorithms satisfy the given service objectives would indicate the efficiency with which the algorithms dispatch the taxis. These service objectives will affect how the dispatching algorithms are optimised and how they are compared with one another. For this thesis, only the following objectives will be considered:

- Minimize the average waiting time of the customer on a request-for-service call (that is the average time taken for a taxi to respond to a call).
- Maximise the average chartered distance of the taxi fleet with respect to the average total distance travelled per taxi. This ratio will be referred to as the Chartered Distance Ratio for the remainder of this report. Maximising this ratio will maximise the revenue of the taxi company as the company only earns revenue on the distance travelled while the taxi is chartered and it spends money (e.g. due to fuel
consumption, wear of vehicles, etc.) running the taxi regardless of whether it is chartered or idle when moving. This ratio can be used to indicate the relative costs of various dispatch algorithms. The lower the ratio, the higher the cost associated with a dispatch algorithm. However, it must be noted that this ratio does not take into account the cost of idle taxi time when it is not moving (which is important if you are paying the taxi drivers per hour). It only takes into account costs associated with operating the taxi while it is moving.

Several other service objectives (such as evenly distributing the load across the fleet of taxis) could be considered if desired.

### 4.1.5 Structure and Programming Details of the Model

This section is concerned with the programming details of the model.

#### 4.1.5.1 Programming language and Software Required

The model was programmed in Python version 2.5.1 on Macintosh OS X 10.4. Python is an interpreted, object-oriented programming language. The reasons for choosing Python for the development of this model are:

- Python combines the power of a higher level programming language with easy to write and understand syntax
- Python is portable and runs on many varieties of operating systems
- Python can be written rapidly and is thus suitable for models that constantly need to be changed. Since the speed at which the simulation runs is not an issue, python is a better choice than C/C++ in terms of ability to rapidly develop code

We must note that Python lacks the built-in visualisation features (which are present in other packages such as MATLAB). However, the class implementation in
Python is very simple and hence Python will be a good choice when considering overall programmability. Visualisation features can be added through the use of separate modules for Python if required in the future.

Modularity was implemented through the use of classes. Each module is a separate class in Python. Although Python lacks features for implementing public and private members and methods, all access to the class is done through access methods (i.e. other classes will not directly change variable values in a class). Methods that are members of the given class are able to change members of that class directly.

4.1.5.2 Simulation Inputs, Parameters, and Outputs

All parameters of the simulation run (number of taxis, taxi speed, etc.) are set by the user prior to run-time. The demand patterns for the simulation are provided by the Grid Module. Incoming calls will be generated according to a set of rules predefined in the code of the Call Event Generator. The rules will define call probabilities by geographic zone and time.

The output data is reported in text format printed on the screen. The required graphs are graphed using an external graphing package – Microsoft Excel was used for this thesis report. The model is capable of outputting all parameters relevant to the simulation as well as calculating customer wait times, chartered distances of taxis, etc. For future development, a graphing library for python, such as Matplotlib [xiv], can be implemented to automate graphs of important simulation outputs (such as vehicle locations, average call wait times, etc.).
4.1.5.3 Programming Logic and Outline

The code for the simulation is shown in Appendix A: Source Code. It is fully commented and is explained reasonably clearly. This section will step through the code to explain in more detail how the model functions. First, each of the modules in the model is explained in more detail below.

4.1.5.3.1 Structure of Modules used in the Model

The modules used generally consist of several functions of which the important ones are described in more detail. As mentioned previously, each module is a separate class. These classes are stored in separate python files (these files have a .py extension) and are called from the main simulation module.

Main Simulation Module (from file taxi_simulation.py)
- This module runs the simulation and calls other modules as required. The simulation itself is described chronologically later. This module Passes information between other modules.
- The module keeps track of simulation time for the system and generates random time intervals for events to occur

Grid Module (Grid Class from file class_grid.py)
This class performs the following roles:
- Represents the city in which the taxis travel
- Contains dimensions of the city
- Contains information on specific geographic zones (downtown, suburbs, etc.)
• Contains information on demand patterns (by location, time of day, etc)

These roles are accomplished by certain member function as described below:

• **divide_grid**: This function divides the grid into 4 sectors – a downtown sector and 4 suburbs. It stores the locations of the sectors as well as the relative demands of each sector in a list. The list is automatically sorted in decreasing order of demand.

• **find_sector**: This function returns which sector a given set of coordinates is in. It does this by searching through the list of sectors and checking whether the coordinates are that sector.

• **random_sector_location**: This function generates a set of random coordinates in a given sector.

• **weighted_sector_selection**: This function randomly selects a sector. The selection is weighted on the relative demand probabilities of all the sectors. A sector with a higher demand will have more of a chance of being selected than a sector with a lower demand. A flag can be included to tell the function whether to include the downtown sector or not (this is used when selecting sectors during rush hours).

**Taxi Module (Taxi Class from file class_taxi.py)**

This class performs the following roles:

• Represents the service vehicle (the taxi) being deployed

• Contains information such as speed, direction, location, status, distance chartered, and total distance travelled of the taxi
- Contains mechanism to move taxis between two points. During the move, statistics are calculated, including distance travelled and estimated time left for the move to be completed.

The following are some notable member functions of this class:

- *calculate_time_left*: This function calculates the time left for the taxi to complete its move by finding the distance left between its current location and its target/destination and dividing it by the speed of the taxi (which is constant).

- *calculate_direction*: this function calculates the angle in radians which the taxi is travelling in, with respect to the bottom left of the grid (which is the origin of the grid). It does this using trigonometric formulas.

- *move*: This function is responsible for actually moving the taxi for a given time interval. It calculates the distances moved in the x and y directions (using the *calculate_direction* function) and then finds the position of the taxi at the end of the time interval using the x and y distances travelled. The function also updates taxi statuses once it reaches its target.

**Call-Generator Module (Call_Event_Generator Class from file class_call_event_generator.py)**

This class performs the following roles:

- The module generates requests for service (calls) by passengers. It generates both caller locations and destination locations.

- The call origin and destination locations depend on the demand distribution throughout the city (i.e. different sectors have different demands at different times).
These are accomplished by the member function described below:

- **generate_call**: This function decides whether an incoming call is present (and generates it if it is present). It decides which call generation method to use based on whether the system is in rush hour or not. If an incoming call is generated, the origin and destination coordinates are returned. If not, then (-1, -1) is returned as the origin coordinate. If an incoming call is present, the function then checks if the call is in the direction of the rush hour traffic. If so, the function calls the appropriate coordinate generation function to generate origin and destination coordinates. If the system is in an incoming rush hour, then the *generate_rush_call_in* function is used. If it is an outgoing rush hour, then the *generate_rush_call_out* is used. Finally, if the sector is not in an incoming or outgoing rush hour, or the call is not in the rush hour direction, then the *generate_sector_call* function is used.

- **generate_sector_call**: This function uses the *weighted_sector_selection* function in the grid module to decide on origin and destination sectors of the call based on the weight. The higher the demand probability for a given sector, the higher the probability of a call starting or ending in that sector.

- **generate_rush_call_in**: This function selects the downtown sector as the destination of the call. One of the suburbs is selected as the origin of the call using the *weighted_sector_selection* in the grid module (a flag is used to ensure that the downtown sector is not selected).

- **generate_rush_call_out**: This function selects the downtown sector as the origin of the call. One of the suburbs is selected as the destination of the call using the
weighted_sector_selection in the grid module (a flag is used to ensure that the downtown sector is not selected).

**Automated Dispatch Module (Auto Dispatch Class from the file class_auto_dispatch.py)**

This class performs the following roles:

- This module decides how the taxis are assigned throughout the grid. It controls how calls are answered and the idle (un-chartered) movements of taxis
- It implements a queuing system to hold calls
- The module uses dispatching rules/algorithms, which can be interchanged in order to compare the relative performance of the different modules. More details on the specific algorithms implemented in this module will be discussed in Section 4.2: Dispatch Algorithm Development of this chapter.

The main functions in this class are:

- **assign_answered**: This function checks which taxis have reached their call origins and assigns the destination coordinates to them. When a call is answered, it is removed from the queue and added into a list of answered calls which is used for statistics purposes.
- **taxi_update**: This function updates the current simulation time and taxi information to the automated dispatch module from the main simulation module.
- **assign_taxis**: This function calls the dispatching rules to decide which taxis are assigned to calls and to decide where idle taxis are to go. The user can create their own dispatching algorithms in the form of functions and call them from this function
in order to implement them. For this thesis, the dispatch algorithms are implemented through the following functions (more details on the algorithm can be found in Section 4.2: Dispatch Algorithm Development of this chapter):

- auto_dispatch_1: implements the random dispatch algorithm
- auto_dispatch_3: implements the vehicle anticipation algorithm

4.1.5.3.2 Break-Down of Simulation Steps

The main simulation module runs the simulation and calls other modules to perform their roles. It also transfers information between modules. In order to illustrate how the simulation runs, the steps in the simulation are broken down in chronological order as shown below. It must be noted that all these steps occur in the main module.

Simulation Steps in the Main Simulation Module:

1. Objects are created for each module (each module is a class). This includes the grid module, the call generator module, the automated dispatch module and the taxi module. The taxis are organised into an array of taxi objects. Their starting locations are randomly generated throughout the grid. The taxi locations, grid, and starting simulation time are passed to the automated dispatch module.

2. The simulation is started at the starting time

3. The time is passed to the call generator module which checks if an incoming call has occurred. The call generator module will return a negative number for the call origin coordinates if not call is present. Otherwise, it generates a call (using the generate_call function) and returns the call origin and destination to the main module. The main module checks whether a call came in and if so, it adds the call details to the queue of the automated dispatch module.
4. The main module updates the current time and taxi locations to the dispatcher (using the `taxi_update` function). Then the dispatcher returns the taxi assignments for each taxi to the main module. How the taxi assignments are determined depends on the algorithm used (see Section 4.2: Dispatch Algorithm Development).

5. The main module then creates a time interval of a random length. These random time intervals are meant to simulate calls arriving at random times. A maximum time interval is used to ensure that there are a given minimum number of calls during each simulation period. The maximum time interval length also increases and decreases depending on the time of the day. The main module checks whether the system is in rush hour or not. If the system is in rush hour, then the maximum time interval is shortened to make the frequency of incoming calls increase during rush hours.

   a. Note that if the length of the time interval is longer than the remain time left in the simulation, then the length of time left in the simulation is used as the time interval instead of the randomly generated number. This is to ensure that the simulation stops at the end time of the simulation.

6. The main module calculates the time left for each taxi to reach its target coordinates. This is done using the `calculate_time_left` function in the taxi module. The main module then finds the shortest time left and compares it with the length of the time interval.

7. The main module increments the current time of the system by the smallest value of the least time left of the taxis or the time interval (i.e. if the time interval is smaller than the least time left for any taxi, then the system time is advanced by the time
interval amount, otherwise the system time is advanced by the least time left of the taxis).

a. All taxis are then moved towards their destinations by the amount of time the system time is incremented by.

b. The new locations of the taxis are then updated on the dispatch module using the \textit{taxi\_update} function. The dispatch module then makes changes to the assignment of taxis as necessary and these are relayed to the taxis.

c. The length of time left in the time interval is then calculated and updated

8. Steps 6-7 are repeated till there is no time left in the given time interval (the length of time left in the time interval is 0).

9. Steps 2-8 are repeated till the current time of the system is equal to the end time of the system.

10. Statistics of the simulation are then printed on screen. The average waiting time is calculated by dividing the total waiting time of all calls answered by the number of calls answered. The average distance chartered is calculated by dividing the total chartered distance of each of the taxis by the number of taxis, and the average total distance travelled is calculated by summing the total distances travelled of each taxi and dividing them by the number of taxis.

It would be beneficial to refer to the flow chart of the entire process presented in \textbf{Figure 7 – Flowchart of Simulation Steps} in the \textbf{Figures and Tables} chapter of this report.
4.2 Dispatch Algorithm Development

The dispatch algorithm provides two important functions – it assigns requests for service (calls) to taxis and it tells idle (unoccupied) taxis how to behave. Two dispatching algorithms were used in this thesis. The first algorithm uses vehicle anticipation to predict where the next call will come from and tries to ensure that a vacant taxi is available nearby (we will refer to this as vehicle anticipation dispatch). The second was a “baseline” algorithm, where no vehicle anticipation is used (we will refer to this as random dispatch). This section details how the two dispatch algorithms work and how they are implemented in the dispatch module.

4.2.1 General Approach

Some general details of the dispatch algorithm methodology are provided in this sub-section.

4.2.1.1 Implementation of the Dispatch Algorithm

As mentioned in the programming details section, the dispatch module is a separate class. This class is given up-to-date information on the locations of all vehicles, their statuses, and the locations of service requests every time it is called. The dispatch module is called anytime an event occurs (i.e. a service request, a pickup, a drop off, etc) and then re-evaluates the assignments of all vehicles (whether they are idle assignments or assignments to service requests).

The dispatch module decides how calls are assigned and assigns coordinates for idle taxis to go, based on the dispatching algorithm. The dispatching algorithm is interchangeable within the dispatch module to allow multiple algorithms to be used.
Furthermore, the dispatching algorithm is divided into two components: one part determines how calls are assigned to taxis, and the other determines where idle taxis will move to. These two components are independent of each other and thus can be changed to allow for various combinations of dispatch algorithms.

The dispatching algorithm is implemented as a separate function within the dispatching module. This function must be called from the \textit{assign\_taxis} function. The random dispatch algorithm is written in the function \textit{auto\_dispatch\_1} while the vehicle anticipation algorithm is implemented in the function \textit{auto\_dispatch\_3}. The dispatch algorithm function can access all taxi locations, taxi statuses, and call information directly from the dispatch module class. The function does not return any values; rather it must then make relevant changes to the taxi targets in the taxi object present in the dispatch module. These changes will then be passed to the main simulation module that will update the target coordinates of the taxis.

\textbf{4.2.1.2 Use of Hierarchical Agents}

The dispatch module will rely on information provided by several agents, which are the individual modules in our case. The dispatch module uses the taxi module (to provide vehicle locations and statuses) and the grid module (to provide information on demand patterns). The arrangement of the agents is hierarchical (as discussed in the literature review section) and all information is passed to the Dispatch agent (which controls overall dispatching). No collaborative features will be implemented and hence individual taxi agents will not decide their assignments or where they will travel when idle.
4.2.2 Baseline Algorithm - Random Dispatch

This dispatch algorithm represents the baseline method of taxi dispatch to which the performance of other dispatch algorithms can be compared. Details of this algorithm are given below:

- **Call Assignment:** The closest available (idle) taxi to a call is assigned to the call. Calls are queued based on the time they come in and are answered on a first-come first-served basis.

- **Idle Taxi Movement:** The random dispatch algorithm does not assign anywhere for idle taxis to go. It lets them remain at their current locations till they are assigned to answer a call.

This dispatching algorithm is the base case against which the Vehicle Anticipation algorithm will be evaluated.

4.2.3 Vehicle Anticipation Dispatch

This algorithm uses vehicle anticipation to determine where the next taxi will likely be needed. Vehicle anticipation is only used when deciding where idle taxis will move. The details of the algorithm are given below:

- **Call Assignment:** As with the random dispatch algorithm, the closest available (idle) taxi is assigned to a given call. Calls are queued based on the time they come in and are answered on a first-come first-served basis.

- **Idle Taxi Movement:** Vehicle anticipation is used to determine where the next taxi will be needed and will move idle an idle taxi to the area the next call is likely to come from. Vehicle anticipation is implemented based on sectors. When a vehicle is required in a sector, it will move to the centre of the sector. The anticipated need for
vehicles is based on demand probabilities of the given sector. The vehicle anticipation implementation also takes rush hours into consideration. Further details are given in the following section.

4.2.3.1 Implementation of Vehicle Anticipation

The vehicle anticipation algorithm determines the number of taxis needed in each sector and then tries to move the required number taxis. During the incoming rush hour, the majority of the taxis will be needed in the suburb sector and hence the vehicle anticipation algorithm tries to move the required number of taxis to the suburbs starting 0.5 hours before the start of the incoming rush hour. The taxis are distributed amongst the suburbs based on the demand probability of each of the suburbs (suburbs with a higher demand probability will have a greater number of taxis needed). Similarly, during the outgoing rush hour, the majority of calls will originate in the downtown sector. Hence the vehicle anticipation algorithm tries to move the required number of taxis to the downtown sector starting 0.5 hours before the outgoing rush hour starts. When the system is outside of either rush hour, the calls originating in each sector will be proportional to the demand in the sector, and hence the vehicle anticipation algorithm distributes the taxis over all sectors proportional to demand. When a taxi is sent to a given sector, it is sent to the centre of the sector.

To calculate the number of taxis needed in each sector, the demand probability of the sector is found. This demand probability is normalised by the sum of the demand probabilities in all the sectors in order to find the relative demand probability of all the sectors. Finally, the relative demand probability of each of the sectors is multiplied by the total number of taxis to determine the number of taxis needed in each sector. Since the
result of this multiplication might yield a floating point value, the product is then floored (rounded down) to the next lowest integer (as taxis are available in integer quantities).

Once the number of taxis needed in each sector is found, the algorithm evaluates the number of occupied (chartered) taxis currently in each sector and subtracts this value from the number of taxis needed in this sector. The algorithm then cycles through all sectors with outstanding need and assigns the closest idle taxi to the sector centre to go there. This happens till all needs are satisfied or till all idle taxis have been assigned to a sector. Note that once a taxi is assigned to a sector, the algorithm moves to the next sector to next sector and keeps looping through the sectors till all the taxis are assigned. This ensures that the taxis are more evenly distributed throughout the sectors.

### 4.3 Algorithm Simulation and Evaluation

An overview of how the dispatching algorithms were simulated and evaluated is given in this section. The results and analysis of the simulation runs is discussed in Chapter 5: Simulation Results and Analysis.

#### 4.3.1 Simulation of inputs

The inputs of the simulation are the incoming calls (requests for service). The frequency of calls is determined based on the length of each time interval as well as the probability of a call coming in at any given time interval. The length of any given time interval is given by a random uniform distribution between zero and the maximum time interval length. The call frequency is varied during rush hours by decreasing the maximum time interval, and hence increasing the number of time intervals that occur over a period of time.
The origin and destination of calls are based on probabilities that a call will originate and terminate in given sectors. First the origin sector and destination sectors are selected based on the demand probabilities for the sectors and depending on whether the system is in rush hour or not. When the system is in rush hour, the majority of the calls will start and terminate in the direction of the rush hour traffic. When the system is not in rush hour, the probability of a call originating or terminating in a sector is found by dividing the demand probability of the sector by the sum of the demand probabilities of all the sectors in the grid. Within each sector, the exact locations calls will start or end is selected randomly (i.e. once it is decided that a call will start or end in a given sector, a random location in the sector is generated). Thus within a given sector, the demand probability is constant.

### 4.3.2 Simulation Outputs

The service objectives being considered are the average call wait time and the average chartered distance ratio per taxi. Hence these are the parameters outputted by each run of the simulation. In a given run, the simulation will average the wait times over all the calls answered in the run output the result. Similarly, the distances chartered and total distances travelled will be averaged over all the taxis in the given run to calculate the average chartered distance ratio.

### 4.3.3 Variation of Inputs and Simulation Parameters

While a full-scale sensitivity analysis was not performed, a few key parameters were varied to determine their effects on the output. The parameters varied were:

- **Number of taxis**: 5, 10, 20, 40, and 80 taxis were used for each simulation case
• **Size of the city:** The length and the width of the city were doubled.

• **Call Probability:** The probability that a call will occur in any given time interval was doubled.

### 4.3.4 Simulation Specifics

Three simulation cases were tested to determine the performance of the vehicle anticipation dispatch algorithm against the random dispatch algorithm. A full listing of the simulation parameters used for each of these cases can be found in Table 13 – *Simulation Parameters for Base Case, Double Frequency Case, and Double Grid Size Case* in the *Figures and Tables* chapter of this thesis report.

The base case was used with parameter values thought to be similar to the City of Toronto. In the second case, the call probability was doubled. This doubled the frequency of calls throughout the simulation run. All other parameters were kept identical to the base case. In the third simulation run, the length and width of the grid was doubled. Again, all other parameters, including call probability were kept identical to the base case.

In all the three simulation cases above, the number of taxis was varied. 5, 10, 20, 40, and 80 taxis were used for each case. Each simulation was run using the random dispatch algorithm and the vehicle anticipation algorithm to determine the relative performance of the algorithms. For each specific simulation, 20 trials were run to increase the accuracy of the output.
5 Simulation Results and Analysis

This section describes the results obtained from the simulation runs conducted and analyses them. It must be noted that these results and the following analysis are specific for the simulation parameters discussed in the previous section.

5.1 Data Processing

As mentioned previously, three simulation cases were conducted: a base case, a case with double the call probability than the base case, and a case with double the grid dimensions of the base case. Each of these cases was run with the random dispatch algorithm and with the vehicle anticipation dispatch algorithm for 5, 10, 20, 40, and 80 taxis. For each trial, the output results were the average customer waiting time and the average chartered distance ratio of the taxis. These values are the service objectives against which the algorithms are being evaluated. 20 trials were conducted of each combination of simulation case, taxi number, and dispatch algorithm used. For each set of 20 trials, the average waiting time and the average distance chartered ratio was averaged amongst the 20 trials. The standard deviation of the samples was calculated and the 95% confidence interval for the mean of the set of samples was found. It was assumed that the output was from a normally distributed population, where the mean and the standard deviation of the population are unknown. The results for each case were then plotted against the number of taxis for each set. The y-error bars represent the 95% confidence interval for the mean. The results for both the random dispatch and the vehicle anticipation dispatch were plotted on the same set of axes to facilitate comparison of the two dispatching algorithms. Figures 1, 2, and 3, in the Figures and Tables chapter of
this thesis report, are graphs of the average waiting time versus the number of taxis for both dispatching algorithms for each simulation case. Figures 4, 5, and 6, in the Figures and Tables chapter, are graphs of the average chartered distance ratio versus the number of taxis for both dispatching algorithms for each simulation case. The average waiting time and average chartered ratio data for each simulation case for each dispatching algorithm is presented in Tables 1 to Table 12 in the Figures and Tables chapter.

5.2 Average Customer Waiting Time

In all cases, we see that regardless of the dispatch algorithm used, the average wait time of the customer decreases with the number of taxis. This was expected as the larger the number of taxis, the more likely there will be an available one close to any given call location. Similarly, the variation in wait times, as represented by the 95% confidence interval and the sample standard deviation, decreases as the number of taxis increase.

5.2.1 Base Case

The graph for the average customer wait times as a function of the number of taxis for both dispatch algorithms in the base case is shown in Figure 1 - Average Customer Waiting Time versus Number of Taxis for Base Case. As can be seen from the graph as well as the tabulated data for the base case, vehicle anticipation leads to a reduction in wait time of the order of approximately 8% for 5 taxis. This reduction in wait time due to the use of vehicle anticipation increases to its maximum of a more than 20% reduction in wait time with around 15 taxis. At 20 taxis, we see that vehicle anticipation and random dispatch yield approximately the same average wait time. Beyond 20 taxis, the random
dispatch algorithm actually yields shorter waiting times than vehicle anticipation (and the difference in wait times increases further as the number of taxis increases).

We must note that the standard deviation of the waiting times is generally less when using vehicle anticipation, unless the number of taxis is very large or very small. Hence the wait times are generally more predictable when using vehicle anticipation, than when using random dispatch.

5.2.2 Effect of Increase in Frequency

The graph of average customer wait times for the double call frequency case is shown in Figure 2 - Average Customer Waiting Time versus Number of Taxis for Double Call Frequency Case. The frequency of calls is double that of the base case. As expected, increasing the frequency of calls increases the wait times for customers when compared to the base case, regardless of the number taxis used. However, we must note that the difference in wait times between using vehicle anticipation and using random dispatch is smaller in most cases. The maximum decrease in wait time due to the use of vehicle anticipation over random dispatch has moved to approximately 25 taxis, from the optimum point of approximately 15 taxis in the base case. As with the base case, as the number of taxis gets larger, random dispatch results in shorter waiting times then when using vehicle anticipation, though to a lesser extent in this case. We also find that the variation of the average wait time is similar to the base case for both vehicle anticipation and random dispatch.
5.2.3 Effect of Increase in Grid Size

The dimensions of the grid (the city) were doubled. The graph of average customer wait times for this case is shown in **Figure 3 - Average Customer Waiting Time versus Number of Taxis for Double Grid Size Case**. As with increasing frequency, an increase in grid size results in longer average waiting times for customers. Again, we see that the differences in waiting time between random dispatch and vehicle anticipation dispatch are smaller than in the base case. Vehicle anticipation yields a maximum decrease in wait time when compared with random dispatch when the number of taxis is approximately 20. Like in the case where the frequency of calls is increased, we can see that this optimal point (where vehicle anticipation yields the lowest reduction in waiting time over random dispatch) is shifted to a higher number of taxis. The variation in average waiting times is approximately double that of the base case, as is indicated by the standard deviation of the average waiting times.

5.2.4 Further Analysis of Average Waiting Time Results

As we can see from the simulation cases, the average waiting times with vehicle anticipation and random dispatch are approximately the same with lower numbers of taxis (around 5-10 taxis). As the number of taxis increases, vehicle anticipation yields lower waiting times than random dispatch till an optimum point (where vehicle anticipation yields the best improvement over random dispatch), after which the waiting times for vehicle anticipation begin to level out (i.e. there is not much significant reduction in waiting times as more taxis are added). In the case of random dispatch however, the average waiting time continues to decrease as more taxis are used. The
result is that as the number of taxis increases, random dispatch yields lower waiting times than vehicle anticipation.

The reason behind this is that as the number of taxis increases and the call frequency remains the same, the taxis are no longer the major constraint on the system and are hence used less. In random dispatch, the taxis are distributed randomly (but somewhat uniformly) over the entire grid initially, and over time, the distribution of taxis will change depending on demand patterns of different sectors. However, since there are so many taxis in the city, each sector will still have sufficient taxis that are distributed throughout the sector to answer calls, and hence it is likely that there will be an available taxi closer to a call when using random dispatch, with a large number of taxis. When using the vehicle anticipation algorithm, the taxis will go to the centre of the respective sectors they were assigned to and hence are not spread out over the sector, resulting in longer waiting times than random dispatch for a large number of taxis (as taxis still need to travel from the centre of the sector to where the call originated from, resulting in little change in wait times as the number of taxis increases). This reasoning is further corroborated by the fact that the differences in wait times decrease as the frequency of calls increases and the grid size increases (the taxis have a higher demand load on them in these cases and hence are not assigned by the vehicle anticipation algorithm to go to the centre of a sector which needs the taxis as often).

5.3 Average Chartered Distance Ratio

The average chartered distance ratio is an indicator of the revenue generated by the taxi company. This can be used to indirectly indicate the costs of various dispatch algorithms. The lower the ratio, the higher the cost associated with a dispatch algorithm.
However, it must be noted that this ratio does not take into account the cost of idle taxi time. The graph of average chartered distance ratio for the base case is shown in Figure 4 – Chartered Distance Ratio Versus Number of Taxis for the Base Case. The graph of average chartered distance ratio for the double call frequency case is shown in Figure 5 - Chartered Distance Ratio Versus Number of Taxis for the Double Call Frequency Case. The graph of average chartered distance ratio for the double grid dimensions case is shown in Figure 6 - Chartered Distance Ratio Versus Number of Taxis for the Double Grid Size Case.

5.3.1 General Trends

As can be seen from the graphs, in all simulation cases, the chartered distance ratio when using random dispatch increases as the number of taxis increases. Similar to the reasoning for average waiting time, this due to the fact that the more taxis there are when using random dispatch, the more distributed they are throughout the grid, and hence the closer they are likely to be to a call (and hence they will have to travel less to a call). It must be noted that the rate of increase in the ratio initially starts off small, and then increases at a faster rate till around 40 taxis. After this point, the ratio increases at a lower rate. From the graphs, it appears that the ratio has an inverse exponential decay relationship with the number of taxis. It appears that the ratio will become horizontally asymptotic at a value of 1.0 as the number of taxis increases.

With vehicle anticipation, we see the reverse happening. As the number of taxis increases, the chartered distance ratio decreases. This is true for all the simulation cases. The reason for this is that the vehicle anticipation algorithm tells idle taxis to go to the centre of the sectors they are needed in (in the random dispatch algorithm, the taxis just
remain wherever they are). This results in more idle (un-chartered) travelling for the taxis than in random dispatch, though in many cases it will shorten the average customer wait time. From the graphs, it appears that the ratio is linearly inversely proportional to the number of taxis when using vehicle anticipation.

5.3.2 Effect of Increasing Frequency of Calls and Grid Size

We can see that increasing grid size and call frequency has little effect on the chartered distance ratio, regardless of the dispatch algorithm used. This is true regardless of the number of taxis used. In the case of the random dispatch algorithm, when the grid size is increased, taxis will generally need to travel more to reach the call locations, as these will likely be further apart on the grid. Similarly, the destination locations will increase proportionally as well. The result is that the ratio between the average chartered distance travelled and the average total distance travelled (which is the chartered distance ratio) will remain constant. Similarly, an increase in call frequency will result in more calls per unit time, however, the average distance between the calls and the taxis will remain the same, as will the average distance between the call origins and call destinations. Thus the chartered distance ratio will remain more or less the same.

The same is generally true when the vehicle anticipation dispatch algorithm is used, however, increasing the call frequency or increasing the grid size increases the chartered distance ratio by a small amount, regardless of the numbers of taxis used. The reason for this is that increasing the call frequency or the grid size will result in taxis being idle less, and hence they will travel to the centre of the sector they are needed in less (as the demand on the taxis is more and hence they have less time to move idle in these calls). However, the increase in the ratio is very small.
5.4 Further Analysis

The chartered distance ratio is lower when using vehicle anticipation than when using random dispatch. This is true regardless of the number of taxis used and the simulation case. This can again be attributed to the fact that the taxis move to the centres of the sectors they are needed when they are idle, resulting in more idle travelling than in the case of random dispatch. It will cost more for taxi companies to operate taxis when using the vehicle anticipation algorithm, as the taxis are earning less per unit distance they travel. It is hoped that this extra travelling would lead to lower customer wait times, however, we see that this is not true in all cases. The vehicle anticipation algorithm leads to lower average wait times than the random dispatch algorithm only when a relatively small number of taxis in relation to the demand are used. Hence for these cases, there is a trade-off between lower average waiting times and cost of operating the taxis using the vehicle anticipation algorithm. As mentioned in the introduction chapter, lower customer wait times result in a more reliable service for the consumer, and customers might be willing to pay more for the added reliability. However, a full cost benefit analysis must be employed to determine whether to use the vehicle anticipation algorithm.

When the number of taxis employed is large in comparison with the demand, the random dispatching algorithm results in both lower average customer wait times, as well as lower average operating costs. Hence this algorithm should be selected over the vehicle anticipation algorithm in this case.

Increasing the call frequency or the grid size puts more strain on the dispatching system (as taxis are called more frequently), and we can see that the differences between the random dispatch algorithm and the vehicle anticipation algorithm decrease than with
the base case. This is likely due to the fact that both dispatch algorithms assign the closest taxi to the call when deciding assignments. Only the idle taxi movements differ between the algorithms. Hence as the demand on the dispatching system increases, there is less idle taxi movement and hence the differences between the algorithms decrease.
6 Conclusion and Future Development

This thesis was primarily focused on developing a computer-based model to simulate a realistic taxi-dispatching environment and developing an algorithm involving vehicle anticipation to dispatch taxis. Overall, I believe the thesis achieves these objectives reasonably well, however there are areas for improvement and further development.

6.1 Development of the Taxi-Dispatching Environment

The computer model developed to simulate the dispatching environment was successful in replicating most of the major environmental considerations in a typical taxi-dispatching scenario. The model is designed to act as a preliminary test bed on which to implement different dispatching algorithms and evaluate their relative performances, which it does successfully. The modular design of the model ensures that different components can be upgraded and/or replaced as necessary. This modularity also allows different dispatching algorithms to be used on the same framework (as only the dispatch module needs to be modified). While the dispatching environment model was designed taking only taxi dispatching into consideration, it can be modified to provide a suitable dispatching environment for many different types of service vehicle dispatch problems.

Given more time, several further improvements can be made to the model to include more realistic elements. These improvements include:

- Developing a road system on which taxis must travel, rather than allowing them to travel in a straight line through the grid. This however would make each model very specific to a given city.
Disturbances in traffic, such as traffic lights, road closures, etc. could be taken into account to yield more accurate results.

Allowing taxis to be picked up of the “street” as they travel idle.

Implementing a graphing module to display graphs of various output values within the dispatch environment would improve user friendliness of the module and save time when testing algorithms.

Overall, the dispatch environment model successfully achieves its objectives and can be used as a preliminary “test-bed” for service vehicle dispatching algorithms. Though the simulation does not provide a complete solution for testing dispatch algorithms, it does provide an indication of the relative performance of different dispatching algorithms and can be used as a tool to screen for promising solutions.

6.2 Vehicle Anticipation Algorithm

The vehicle anticipation algorithm did not perform as expected in the dispatching environment. It was hoped this implementation of vehicle anticipation would result in significantly lower average customer waiting times over a wide range of fleet sizes, with little additional cost. It was found that this was not the case, as outlined in Chapter 5: Simulation Results and Analysis. The implementation of vehicle anticipation only lowered average customer waiting times (compared to the random dispatch algorithm) for a small range of fleet sizes, numbering between 15-25 taxis. Furthermore, vehicle anticipation did result in higher costs than random dispatch and hence a full analysis must be performed to determine whether the reduced wait times are worth the additional cost in these cases. This analysis is outside the scope of this thesis. When the fleet size is large compared to the demand and the grid area, the random dispatch algorithm shows
significant benefits over the current implementation of vehicle anticipation. It results in lower average waiting times and reduced costs and hence should be implemented in this case. Increasing the demand load on taxis, through increasing the frequency of incoming calls and increasing the grid size, narrowed the differences between the vehicle anticipation algorithm and the random dispatch algorithm. The reason behind this is that once the demand load on the taxis increases, dispatching depends less on anticipation due to the decreased idle time of the taxis. The taxis are assigned to the next call sooner and hence are idle less.

The implementation of vehicle anticipation for this thesis was very simplistic. It only took into account demand probabilities for sectors as a whole and assigned the closest available taxi to a call. The queuing system was also rudimentary and assigned calls on a first-come-first serve basis. Several improvements could be made to the implementation of vehicle anticipation, as illustrated below:

- When considering the assignment of taxis to calls, the algorithm only takes into account taxis that are currently available. It could also take into account taxis that are currently occupied to see when they will become available, as these might be able to respond to the given call more quickly.

- Calls could be answered in order of closest distance to a taxi. Average waiting time can be reduced by assigning calls with taxis near them before others, rather than assigning calls in the order they come in. However, this will lead to longer maximal wait times [v]. Calls could move up in priority the longer they wait to reduce the maximal waiting time.
The vehicle anticipation algorithm could make idle taxis patrol given areas to ensure that the entire grid is covered more effectively. This would however increase the cost of running the algorithm, as the chartered distance ratio would decrease.

Overall, the results show that vehicle anticipation does show promise in certain situations, as it lowers the average waiting time in these cases. However, since vehicle anticipation implementations are generally based on heuristics, they must be optimised for the specific dispatching environment used, taking into account all the variables and constraints of the environment.
7 References


[ii] "Taxi and limousine services, operating statistics, by province and territory", [Online document], 2006 Jun. 7, [2007 Nov. 1], Available at: http://www40.statcan.ca/l01/cst01/trad49a.htm


[iv] Leslie Xinru Wang, Simulation on Service Vehicle Dispatching, Toronto, Ont.: Department of Mechanical and Industrial Engineering, University of Toronto, 2005.


[xiv] "Matplotlib / pylab - matlab style python plotting (plots, graphs, charts)", [Product webpage], [2007 Nov. 1], Available at: http://matplotlib.sourceforge.net/
Figure 1 - Average Customer Waiting Time versus Number of Taxis for Base Case
(Note: the error bars are the 95% confidence intervals for the mean of the average call waiting times)
Average Wait Time vs Number of Taxis
(Double Call Frequency)

Figure 2 - Average Customer Waiting Time versus Number of Taxis for Double Call Frequency Case
Figure 3 - Average Customer Waiting Time versus Number of Taxis for Double Grid Size Case
Chartered Distance Ratio vs Number of Taxis (Base Case)

Figure 4 – Chartered Distance Ratio Versus Number of Taxis for the Base Case
Figure 5 - Chartered Distance Ratio Versus Number of Taxis for the Double Call Frequency Case
Figure 6 - Chartered Distance Ratio Versus Number of Taxis for the Double Grid Size Case
Figure 7 – Flowchart of Simulation Steps
Table 1 – Average Customer Waiting Time (hours) using Random Dispatch for Base Case

<table>
<thead>
<tr>
<th>Average Customer Waiting Time (Hours)</th>
<th>Number of Taxis</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>5</td>
</tr>
<tr>
<td>1</td>
<td>1.12248</td>
</tr>
<tr>
<td>2</td>
<td>1.23079</td>
</tr>
<tr>
<td>3</td>
<td>1.26424</td>
</tr>
<tr>
<td>4</td>
<td>1.33573</td>
</tr>
<tr>
<td>5</td>
<td>1.35479</td>
</tr>
<tr>
<td>6</td>
<td>1.55040</td>
</tr>
<tr>
<td>7</td>
<td>1.63285</td>
</tr>
<tr>
<td>8</td>
<td>1.63432</td>
</tr>
<tr>
<td>9</td>
<td>1.63853</td>
</tr>
<tr>
<td>10</td>
<td>1.68838</td>
</tr>
<tr>
<td>11</td>
<td>1.80998</td>
</tr>
<tr>
<td>12</td>
<td>1.88880</td>
</tr>
<tr>
<td>13</td>
<td>2.00000</td>
</tr>
<tr>
<td>14</td>
<td>2.00160</td>
</tr>
<tr>
<td>15</td>
<td>2.46447</td>
</tr>
<tr>
<td>16</td>
<td>2.58215</td>
</tr>
<tr>
<td>17</td>
<td>2.67076</td>
</tr>
<tr>
<td>18</td>
<td>2.96566</td>
</tr>
<tr>
<td>19</td>
<td>2.96813</td>
</tr>
<tr>
<td>20</td>
<td>3.12123</td>
</tr>
<tr>
<td>Average</td>
<td>1.94626</td>
</tr>
<tr>
<td>Standard Deviation</td>
<td>0.63167</td>
</tr>
<tr>
<td>95% Confidence Interval</td>
<td>0.29563</td>
</tr>
</tbody>
</table>
Table 2 – Average Customer Waiting Time (hours) using Vehicle Anticipation for Base Case

<table>
<thead>
<tr>
<th>Average Customer Waiting Time (Hours)</th>
<th>Number of Taxis</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>5</td>
</tr>
<tr>
<td>Trial</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1.03757</td>
</tr>
<tr>
<td>2</td>
<td>1.08095</td>
</tr>
<tr>
<td>3</td>
<td>1.10552</td>
</tr>
<tr>
<td>4</td>
<td>1.20848</td>
</tr>
<tr>
<td>5</td>
<td>1.21543</td>
</tr>
<tr>
<td>6</td>
<td>1.36166</td>
</tr>
<tr>
<td>7</td>
<td>1.39023</td>
</tr>
<tr>
<td>8</td>
<td>1.47232</td>
</tr>
<tr>
<td>9</td>
<td>1.51457</td>
</tr>
<tr>
<td>10</td>
<td>1.53259</td>
</tr>
<tr>
<td>11</td>
<td>1.53817</td>
</tr>
<tr>
<td>12</td>
<td>1.74866</td>
</tr>
<tr>
<td>13</td>
<td>1.75038</td>
</tr>
<tr>
<td>14</td>
<td>2.01040</td>
</tr>
<tr>
<td>15</td>
<td>2.06151</td>
</tr>
<tr>
<td>16</td>
<td>2.17457</td>
</tr>
<tr>
<td>17</td>
<td>2.35715</td>
</tr>
<tr>
<td>18</td>
<td>2.46266</td>
</tr>
<tr>
<td>19</td>
<td>2.69621</td>
</tr>
<tr>
<td>20</td>
<td>3.83956</td>
</tr>
<tr>
<td>Average</td>
<td>1.77793</td>
</tr>
<tr>
<td>Standard Deviation</td>
<td>0.68526</td>
</tr>
<tr>
<td>95% Confidence Interval</td>
<td>0.32071</td>
</tr>
</tbody>
</table>
Table 3 - Average Customer Waiting Time (hours) using Random Dispatch for Double Call Frequency Case

<table>
<thead>
<tr>
<th>Trial</th>
<th>Number of Taxis</th>
<th>5</th>
<th>10</th>
<th>20</th>
<th>40</th>
<th>80</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td>4.49664</td>
<td>0.93363</td>
<td>0.24879</td>
<td>0.12393</td>
<td>0.06526</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>4.64139</td>
<td>1.11903</td>
<td>0.26762</td>
<td>0.13697</td>
<td>0.06699</td>
</tr>
<tr>
<td>3</td>
<td></td>
<td>4.78474</td>
<td>1.18255</td>
<td>0.26966</td>
<td>0.14684</td>
<td>0.06881</td>
</tr>
<tr>
<td>4</td>
<td></td>
<td>4.92926</td>
<td>1.27539</td>
<td>0.27292</td>
<td>0.14889</td>
<td>0.07116</td>
</tr>
<tr>
<td>5</td>
<td></td>
<td>4.98663</td>
<td>1.40865</td>
<td>0.27411</td>
<td>0.15112</td>
<td>0.07334</td>
</tr>
<tr>
<td>6</td>
<td></td>
<td>4.99445</td>
<td>1.46163</td>
<td>0.27490</td>
<td>0.15216</td>
<td>0.07352</td>
</tr>
<tr>
<td>7</td>
<td></td>
<td>5.06727</td>
<td>1.46850</td>
<td>0.27696</td>
<td>0.15279</td>
<td>0.07481</td>
</tr>
<tr>
<td>8</td>
<td></td>
<td>5.16945</td>
<td>1.52588</td>
<td>0.27878</td>
<td>0.15321</td>
<td>0.07679</td>
</tr>
<tr>
<td>9</td>
<td></td>
<td>5.19272</td>
<td>1.53012</td>
<td>0.28111</td>
<td>0.15387</td>
<td>0.07686</td>
</tr>
<tr>
<td>10</td>
<td></td>
<td>5.22600</td>
<td>1.55361</td>
<td>0.28257</td>
<td>0.15431</td>
<td>0.07687</td>
</tr>
<tr>
<td>11</td>
<td></td>
<td>5.32132</td>
<td>1.63301</td>
<td>0.28506</td>
<td>0.15529</td>
<td>0.07814</td>
</tr>
<tr>
<td>12</td>
<td></td>
<td>5.37709</td>
<td>1.72449</td>
<td>0.29384</td>
<td>0.15535</td>
<td>0.07988</td>
</tr>
<tr>
<td>13</td>
<td></td>
<td>5.38827</td>
<td>1.75579</td>
<td>0.29426</td>
<td>0.15844</td>
<td>0.08062</td>
</tr>
<tr>
<td>14</td>
<td></td>
<td>5.40402</td>
<td>1.84081</td>
<td>0.29455</td>
<td>0.15965</td>
<td>0.08111</td>
</tr>
<tr>
<td>15</td>
<td></td>
<td>5.40873</td>
<td>1.84696</td>
<td>0.30611</td>
<td>0.16906</td>
<td>0.08114</td>
</tr>
<tr>
<td>16</td>
<td></td>
<td>5.42836</td>
<td>1.88757</td>
<td>0.30833</td>
<td>0.16912</td>
<td>0.08194</td>
</tr>
<tr>
<td>17</td>
<td></td>
<td>5.69369</td>
<td>1.89626</td>
<td>0.31271</td>
<td>0.17145</td>
<td>0.08263</td>
</tr>
<tr>
<td>18</td>
<td></td>
<td>5.75438</td>
<td>1.91186</td>
<td>0.32090</td>
<td>0.17344</td>
<td>0.08354</td>
</tr>
<tr>
<td>19</td>
<td></td>
<td>6.00772</td>
<td>2.07712</td>
<td>0.33156</td>
<td>0.17365</td>
<td>0.08409</td>
</tr>
<tr>
<td>20</td>
<td></td>
<td>6.30966</td>
<td>2.19077</td>
<td>0.37923</td>
<td>0.17516</td>
<td>0.08701</td>
</tr>
<tr>
<td>Average</td>
<td></td>
<td>5.27909</td>
<td>1.61118</td>
<td>0.29270</td>
<td>0.15673</td>
<td>0.07722</td>
</tr>
<tr>
<td>Standard Deviation</td>
<td></td>
<td>0.44058</td>
<td>0.32786</td>
<td>0.02857</td>
<td>0.01290</td>
<td>0.00596</td>
</tr>
<tr>
<td>95% Confidence Interval</td>
<td></td>
<td>0.20620</td>
<td>0.15344</td>
<td>0.01337</td>
<td>0.00604</td>
<td>0.00279</td>
</tr>
</tbody>
</table>
Table 4 - Average Customer Waiting Time (hours) using Vehicle Anticipation for Double Call Frequency Case

<table>
<thead>
<tr>
<th>Average Customer Waiting Time (Hours)</th>
<th>Number of Taxis</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>5</td>
</tr>
<tr>
<td>1</td>
<td>4.14759</td>
</tr>
<tr>
<td>2</td>
<td>4.18586</td>
</tr>
<tr>
<td>3</td>
<td>4.50247</td>
</tr>
<tr>
<td>4</td>
<td>4.71768</td>
</tr>
<tr>
<td>5</td>
<td>4.86834</td>
</tr>
<tr>
<td>6</td>
<td>5.02997</td>
</tr>
<tr>
<td>7</td>
<td>5.03135</td>
</tr>
<tr>
<td>8</td>
<td>5.03476</td>
</tr>
<tr>
<td>9</td>
<td>5.11019</td>
</tr>
<tr>
<td>10</td>
<td>5.12328</td>
</tr>
<tr>
<td>11</td>
<td>5.17338</td>
</tr>
<tr>
<td>12</td>
<td>5.26599</td>
</tr>
<tr>
<td>13</td>
<td>5.29837</td>
</tr>
<tr>
<td>14</td>
<td>5.36676</td>
</tr>
<tr>
<td>15</td>
<td>5.47416</td>
</tr>
<tr>
<td>16</td>
<td>5.52440</td>
</tr>
<tr>
<td>17</td>
<td>5.53905</td>
</tr>
<tr>
<td>18</td>
<td>5.58792</td>
</tr>
<tr>
<td>19</td>
<td>5.71749</td>
</tr>
<tr>
<td>20</td>
<td>6.06184</td>
</tr>
<tr>
<td>Average</td>
<td>5.13804</td>
</tr>
<tr>
<td>Standard Deviation</td>
<td>0.48591</td>
</tr>
<tr>
<td>95% Confidence Interval</td>
<td>0.22741</td>
</tr>
</tbody>
</table>
### Table 5 - Average Customer Waiting Time (hours) using Random Dispatch for Double Grid Size Case

<table>
<thead>
<tr>
<th>Average Customer Waiting Time (Hours)</th>
<th>Number of Taxis</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>5</td>
</tr>
<tr>
<td><strong>Trial</strong></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>3.70466</td>
</tr>
<tr>
<td>2</td>
<td>4.39547</td>
</tr>
<tr>
<td>3</td>
<td>4.47298</td>
</tr>
<tr>
<td>4</td>
<td>4.72942</td>
</tr>
<tr>
<td>5</td>
<td>4.78683</td>
</tr>
<tr>
<td>6</td>
<td>4.88116</td>
</tr>
<tr>
<td>7</td>
<td>4.96831</td>
</tr>
<tr>
<td>8</td>
<td>5.01287</td>
</tr>
<tr>
<td>9</td>
<td>5.01728</td>
</tr>
<tr>
<td>10</td>
<td>5.21372</td>
</tr>
<tr>
<td>11</td>
<td>5.30201</td>
</tr>
<tr>
<td>12</td>
<td>5.32926</td>
</tr>
<tr>
<td>13</td>
<td>5.36062</td>
</tr>
<tr>
<td>14</td>
<td>5.37095</td>
</tr>
<tr>
<td>15</td>
<td>5.38341</td>
</tr>
<tr>
<td>16</td>
<td>5.39944</td>
</tr>
<tr>
<td>17</td>
<td>5.50706</td>
</tr>
<tr>
<td>18</td>
<td>5.51402</td>
</tr>
<tr>
<td>19</td>
<td>5.58546</td>
</tr>
<tr>
<td>20</td>
<td>5.99315</td>
</tr>
<tr>
<td><strong>Average</strong></td>
<td><strong>5.09640</strong></td>
</tr>
<tr>
<td><strong>Standard Deviation</strong></td>
<td><strong>0.50987</strong></td>
</tr>
<tr>
<td><strong>95% Confidence Interval</strong></td>
<td><strong>0.23862</strong></td>
</tr>
</tbody>
</table>
Table 6 - Average Customer Waiting Time (hours) using Vehicle Anticipation for Double Grid Size Case

<table>
<thead>
<tr>
<th>Average Customer Waiting Time (Hours)</th>
<th>5</th>
<th>10</th>
<th>20</th>
<th>40</th>
<th>80</th>
</tr>
</thead>
<tbody>
<tr>
<td>Trial</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>4.15854</td>
<td>0.70769</td>
<td>0.31100</td>
<td>0.30954</td>
<td>0.30159</td>
</tr>
<tr>
<td>2</td>
<td>4.39550</td>
<td>1.02225</td>
<td>0.32570</td>
<td>0.31032</td>
<td>0.31455</td>
</tr>
<tr>
<td>3</td>
<td>4.71238</td>
<td>1.15841</td>
<td>0.34207</td>
<td>0.31763</td>
<td>0.31564</td>
</tr>
<tr>
<td>4</td>
<td>4.73108</td>
<td>1.29660</td>
<td>0.34350</td>
<td>0.32220</td>
<td>0.31921</td>
</tr>
<tr>
<td>5</td>
<td>4.84616</td>
<td>1.33374</td>
<td>0.35479</td>
<td>0.33394</td>
<td>0.32193</td>
</tr>
<tr>
<td>6</td>
<td>4.94122</td>
<td>1.51050</td>
<td>0.36283</td>
<td>0.33567</td>
<td>0.32240</td>
</tr>
<tr>
<td>7</td>
<td>4.96404</td>
<td>1.54634</td>
<td>0.36518</td>
<td>0.34030</td>
<td>0.32317</td>
</tr>
<tr>
<td>8</td>
<td>4.96737</td>
<td>1.58494</td>
<td>0.36526</td>
<td>0.34591</td>
<td>0.32934</td>
</tr>
<tr>
<td>9</td>
<td>4.98240</td>
<td>1.65204</td>
<td>0.36756</td>
<td>0.34683</td>
<td>0.33192</td>
</tr>
<tr>
<td>10</td>
<td>5.01373</td>
<td>1.65995</td>
<td>0.37179</td>
<td>0.34702</td>
<td>0.33242</td>
</tr>
<tr>
<td>11</td>
<td>5.42533</td>
<td>1.78861</td>
<td>0.37424</td>
<td>0.34835</td>
<td>0.33430</td>
</tr>
<tr>
<td>12</td>
<td>5.77641</td>
<td>1.82199</td>
<td>0.38934</td>
<td>0.35302</td>
<td>0.33501</td>
</tr>
<tr>
<td>13</td>
<td>5.79981</td>
<td>1.87767</td>
<td>0.39757</td>
<td>0.35426</td>
<td>0.33551</td>
</tr>
<tr>
<td>14</td>
<td>5.80792</td>
<td>1.93561</td>
<td>0.40088</td>
<td>0.36017</td>
<td>0.33877</td>
</tr>
<tr>
<td>15</td>
<td>5.82829</td>
<td>2.00002</td>
<td>0.40381</td>
<td>0.36223</td>
<td>0.35433</td>
</tr>
<tr>
<td>16</td>
<td>5.83691</td>
<td>2.05946</td>
<td>0.40791</td>
<td>0.36430</td>
<td>0.35439</td>
</tr>
<tr>
<td>17</td>
<td>5.94951</td>
<td>2.09693</td>
<td>0.41489</td>
<td>0.36677</td>
<td>0.35796</td>
</tr>
<tr>
<td>18</td>
<td>6.03232</td>
<td>2.14573</td>
<td>0.42342</td>
<td>0.36785</td>
<td>0.35883</td>
</tr>
<tr>
<td>19</td>
<td>6.36182</td>
<td>2.79273</td>
<td>0.44649</td>
<td>0.36869</td>
<td>0.36182</td>
</tr>
<tr>
<td>20</td>
<td>6.37107</td>
<td>3.04056</td>
<td>0.45943</td>
<td>0.38231</td>
<td>0.37183</td>
</tr>
<tr>
<td>Average</td>
<td>5.34509</td>
<td>1.75159</td>
<td>0.38138</td>
<td>0.34687</td>
<td>0.33575</td>
</tr>
<tr>
<td>Standard Deviation</td>
<td>0.65055</td>
<td>0.54936</td>
<td>0.03844</td>
<td>0.02040</td>
<td>0.01864</td>
</tr>
<tr>
<td>95% Confidence Interval</td>
<td>0.30446</td>
<td>0.25710</td>
<td>0.01799</td>
<td>0.00955</td>
<td>0.00872</td>
</tr>
</tbody>
</table>
Table 7 - Average Chartered Distance Ratio using Random Dispatch for Base Case

<table>
<thead>
<tr>
<th>Trial</th>
<th>Average Chartered Distance Ratio</th>
<th>Number of Taxis</th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>5</td>
<td>10</td>
<td>20</td>
<td>40</td>
<td>80</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0.52109</td>
<td>0.64378</td>
<td>0.73995</td>
<td>0.85288</td>
<td>0.89557</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>0.49139</td>
<td>0.63972</td>
<td>0.74504</td>
<td>0.84075</td>
<td>0.90015</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>0.53826</td>
<td>0.64255</td>
<td>0.73087</td>
<td>0.83779</td>
<td>0.89753</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>0.51052</td>
<td>0.62142</td>
<td>0.74063</td>
<td>0.83715</td>
<td>0.89972</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>0.53268</td>
<td>0.60900</td>
<td>0.73470</td>
<td>0.82778</td>
<td>0.88651</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>0.51572</td>
<td>0.60645</td>
<td>0.73681</td>
<td>0.83212</td>
<td>0.88795</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>0.52217</td>
<td>0.61324</td>
<td>0.73026</td>
<td>0.81348</td>
<td>0.88683</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>0.49463</td>
<td>0.62191</td>
<td>0.71637</td>
<td>0.82298</td>
<td>0.88292</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>0.51807</td>
<td>0.60188</td>
<td>0.71946</td>
<td>0.82223</td>
<td>0.88555</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>0.50039</td>
<td>0.60701</td>
<td>0.71293</td>
<td>0.82303</td>
<td>0.88908</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>0.48783</td>
<td>0.58968</td>
<td>0.71202</td>
<td>0.81749</td>
<td>0.89096</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>0.50619</td>
<td>0.59756</td>
<td>0.72919</td>
<td>0.82264</td>
<td>0.88332</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>0.50492</td>
<td>0.56938</td>
<td>0.70350</td>
<td>0.81788</td>
<td>0.87756</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>0.52919</td>
<td>0.62215</td>
<td>0.71069</td>
<td>0.81366</td>
<td>0.88260</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>0.52255</td>
<td>0.56455</td>
<td>0.71290</td>
<td>0.82325</td>
<td>0.86997</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>0.54085</td>
<td>0.59568</td>
<td>0.71404</td>
<td>0.81576</td>
<td>0.88036</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>17</td>
<td>0.49070</td>
<td>0.58822</td>
<td>0.69948</td>
<td>0.80713</td>
<td>0.87685</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>18</td>
<td>0.52476</td>
<td>0.57253</td>
<td>0.70175</td>
<td>0.80369</td>
<td>0.86782</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>19</td>
<td>0.51529</td>
<td>0.57164</td>
<td>0.68664</td>
<td>0.80969</td>
<td>0.85807</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>0.52844</td>
<td>0.56095</td>
<td>0.70431</td>
<td>0.78747</td>
<td>0.86130</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Average</strong></td>
<td>0.51478</td>
<td>0.60197</td>
<td>0.71908</td>
<td>0.82144</td>
<td>0.88303</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Standard Deviation</strong></td>
<td>0.01600</td>
<td>0.02571</td>
<td>0.01610</td>
<td>0.01458</td>
<td>0.01185</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>95% Confidence Interval</strong></td>
<td>0.00749</td>
<td>0.01203</td>
<td>0.00754</td>
<td>0.00682</td>
<td>0.00555</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Average Chartered Distance Ratio</td>
<td>5</td>
<td>10</td>
<td>20</td>
<td>40</td>
<td>80</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>----------------------------------</td>
<td>------------</td>
<td>------------</td>
<td>------------</td>
<td>------------</td>
<td>------------</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Trial</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0.49735</td>
<td>0.44485</td>
<td>0.41941</td>
<td>0.36336</td>
<td>0.31166</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>0.46826</td>
<td>0.44725</td>
<td>0.43431</td>
<td>0.36509</td>
<td>0.29339</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>0.46061</td>
<td>0.45250</td>
<td>0.42156</td>
<td>0.37222</td>
<td>0.27824</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>0.48374</td>
<td>0.43965</td>
<td>0.41404</td>
<td>0.36430</td>
<td>0.28109</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>0.48640</td>
<td>0.43702</td>
<td>0.42986</td>
<td>0.37666</td>
<td>0.27965</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>0.50323</td>
<td>0.46565</td>
<td>0.39556</td>
<td>0.37524</td>
<td>0.30609</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>0.48351</td>
<td>0.44245</td>
<td>0.41892</td>
<td>0.36796</td>
<td>0.29212</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>0.47854</td>
<td>0.43756</td>
<td>0.40801</td>
<td>0.37181</td>
<td>0.29177</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>0.48569</td>
<td>0.44100</td>
<td>0.39560</td>
<td>0.36395</td>
<td>0.29425</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>0.48472</td>
<td>0.44908</td>
<td>0.42322</td>
<td>0.37606</td>
<td>0.29250</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>0.49642</td>
<td>0.44488</td>
<td>0.42244</td>
<td>0.36537</td>
<td>0.29116</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>0.46886</td>
<td>0.44239</td>
<td>0.40374</td>
<td>0.38918</td>
<td>0.28271</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>0.47319</td>
<td>0.43881</td>
<td>0.43199</td>
<td>0.36438</td>
<td>0.28931</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>0.49399</td>
<td>0.44276</td>
<td>0.40558</td>
<td>0.34895</td>
<td>0.31327</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>0.50031</td>
<td>0.45175</td>
<td>0.41674</td>
<td>0.35751</td>
<td>0.28770</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>0.45854</td>
<td>0.44465</td>
<td>0.40061</td>
<td>0.34697</td>
<td>0.29463</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>17</td>
<td>0.49641</td>
<td>0.42971</td>
<td>0.39984</td>
<td>0.35316</td>
<td>0.30787</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>18</td>
<td>0.47211</td>
<td>0.43129</td>
<td>0.41642</td>
<td>0.37073</td>
<td>0.26370</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>19</td>
<td>0.49589</td>
<td>0.45486</td>
<td>0.38503</td>
<td>0.36762</td>
<td>0.28091</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>0.49499</td>
<td>0.46447</td>
<td>0.39927</td>
<td>0.35434</td>
<td>0.29160</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Average</td>
<td>0.48414</td>
<td>0.44513</td>
<td>0.41211</td>
<td>0.36574</td>
<td>0.29118</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Standard Deviation</td>
<td>0.01348</td>
<td>0.00937</td>
<td>0.01362</td>
<td>0.01021</td>
<td>0.01211</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>95% Confidence Interval</td>
<td>0.00631</td>
<td>0.00439</td>
<td>0.00637</td>
<td>0.00478</td>
<td>0.00567</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Table 9 - Average Chartered Distance Ratio using Random Dispatch for Double Frequency Case

<table>
<thead>
<tr>
<th>Trial</th>
<th>Number of Taxis</th>
<th>5</th>
<th>10</th>
<th>20</th>
<th>40</th>
<th>80</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>0.51128</td>
<td>0.53053</td>
<td>0.66545</td>
<td>0.79386</td>
<td>0.88014</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>0.50159</td>
<td>0.53723</td>
<td>0.64041</td>
<td>0.78093</td>
<td>0.88025</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>0.50518</td>
<td>0.52743</td>
<td>0.65274</td>
<td>0.75594</td>
<td>0.87096</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>0.49752</td>
<td>0.54137</td>
<td>0.64198</td>
<td>0.76693</td>
<td>0.87669</td>
</tr>
<tr>
<td>5</td>
<td>2</td>
<td>0.48587</td>
<td>0.53682</td>
<td>0.62975</td>
<td>0.75707</td>
<td>0.86282</td>
</tr>
<tr>
<td>6</td>
<td>2</td>
<td>0.52374</td>
<td>0.52804</td>
<td>0.65221</td>
<td>0.76288</td>
<td>0.87012</td>
</tr>
<tr>
<td>7</td>
<td>2</td>
<td>0.51304</td>
<td>0.52603</td>
<td>0.64973</td>
<td>0.76688</td>
<td>0.87145</td>
</tr>
<tr>
<td>8</td>
<td>2</td>
<td>0.51251</td>
<td>0.52794</td>
<td>0.64330</td>
<td>0.76085</td>
<td>0.86911</td>
</tr>
<tr>
<td>9</td>
<td>2</td>
<td>0.50679</td>
<td>0.53166</td>
<td>0.64400</td>
<td>0.76088</td>
<td>0.86607</td>
</tr>
<tr>
<td>10</td>
<td>2</td>
<td>0.46284</td>
<td>0.51368</td>
<td>0.63800</td>
<td>0.75006</td>
<td>0.86773</td>
</tr>
<tr>
<td>11</td>
<td>2</td>
<td>0.50021</td>
<td>0.52446</td>
<td>0.63678</td>
<td>0.75738</td>
<td>0.86727</td>
</tr>
<tr>
<td>12</td>
<td>2</td>
<td>0.51284</td>
<td>0.52854</td>
<td>0.62734</td>
<td>0.76652</td>
<td>0.85786</td>
</tr>
<tr>
<td>13</td>
<td>2</td>
<td>0.49528</td>
<td>0.53131</td>
<td>0.64882</td>
<td>0.76043</td>
<td>0.86354</td>
</tr>
<tr>
<td>14</td>
<td>2</td>
<td>0.48012</td>
<td>0.51596</td>
<td>0.62634</td>
<td>0.75902</td>
<td>0.85777</td>
</tr>
<tr>
<td>15</td>
<td>2</td>
<td>0.50182</td>
<td>0.54282</td>
<td>0.63920</td>
<td>0.74887</td>
<td>0.85398</td>
</tr>
<tr>
<td>16</td>
<td>2</td>
<td>0.49848</td>
<td>0.52532</td>
<td>0.63668</td>
<td>0.73832</td>
<td>0.85296</td>
</tr>
<tr>
<td>17</td>
<td>2</td>
<td>0.48586</td>
<td>0.52498</td>
<td>0.62951</td>
<td>0.73880</td>
<td>0.85612</td>
</tr>
<tr>
<td>18</td>
<td>2</td>
<td>0.51000</td>
<td>0.52424</td>
<td>0.61416</td>
<td>0.73956</td>
<td>0.85256</td>
</tr>
<tr>
<td>19</td>
<td>2</td>
<td>0.49814</td>
<td>0.52294</td>
<td>0.62702</td>
<td>0.73503</td>
<td>0.84889</td>
</tr>
<tr>
<td>20</td>
<td>2</td>
<td>0.50423</td>
<td>0.52804</td>
<td>0.59694</td>
<td>0.73031</td>
<td>0.85334</td>
</tr>
<tr>
<td>Average</td>
<td></td>
<td>0.50037</td>
<td>0.52847</td>
<td>0.63702</td>
<td>0.75653</td>
<td>0.86398</td>
</tr>
<tr>
<td>Standard Deviation</td>
<td></td>
<td>0.01374</td>
<td>0.00729</td>
<td>0.01498</td>
<td>0.01554</td>
<td>0.00951</td>
</tr>
<tr>
<td>95% Confidence Interval</td>
<td></td>
<td>0.00643</td>
<td>0.00341</td>
<td>0.00701</td>
<td>0.00727</td>
<td>0.00445</td>
</tr>
</tbody>
</table>
Table 10 - Average Chartered Distance Ratio using Vehicle Anticipation for Double Frequency Case

<table>
<thead>
<tr>
<th>Trial</th>
<th>Average Chartered Distance Ratio</th>
<th>5</th>
<th>10</th>
<th>20</th>
<th>40</th>
<th>80</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.50854</td>
<td>0.49482</td>
<td>0.46291</td>
<td>0.44975</td>
<td>0.37606</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>0.50869</td>
<td>0.48214</td>
<td>0.46821</td>
<td>0.44070</td>
<td>0.39322</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>0.48566</td>
<td>0.49401</td>
<td>0.46272</td>
<td>0.45669</td>
<td>0.36865</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>0.47677</td>
<td>0.49230</td>
<td>0.45938</td>
<td>0.43471</td>
<td>0.39212</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>0.51075</td>
<td>0.48127</td>
<td>0.45833</td>
<td>0.43389</td>
<td>0.38705</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>0.50933</td>
<td>0.48947</td>
<td>0.45874</td>
<td>0.42725</td>
<td>0.38331</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>0.49643</td>
<td>0.49329</td>
<td>0.49079</td>
<td>0.43061</td>
<td>0.38255</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>0.48313</td>
<td>0.47104</td>
<td>0.47155</td>
<td>0.45385</td>
<td>0.38623</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>0.50810</td>
<td>0.49243</td>
<td>0.45554</td>
<td>0.42670</td>
<td>0.39318</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>0.50266</td>
<td>0.48537</td>
<td>0.45572</td>
<td>0.43869</td>
<td>0.38788</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>0.50371</td>
<td>0.50157</td>
<td>0.47152</td>
<td>0.43720</td>
<td>0.36616</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>0.50109</td>
<td>0.48129</td>
<td>0.45293</td>
<td>0.44150</td>
<td>0.38265</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>0.49455</td>
<td>0.49005</td>
<td>0.46911</td>
<td>0.42677</td>
<td>0.38881</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>0.46967</td>
<td>0.48676</td>
<td>0.47365</td>
<td>0.44130</td>
<td>0.37032</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>0.48905</td>
<td>0.50173</td>
<td>0.46002</td>
<td>0.44673</td>
<td>0.37971</td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>0.50258</td>
<td>0.49813</td>
<td>0.47246</td>
<td>0.44513</td>
<td>0.37906</td>
<td></td>
</tr>
<tr>
<td>17</td>
<td>0.48983</td>
<td>0.48287</td>
<td>0.47207</td>
<td>0.44956</td>
<td>0.39215</td>
<td></td>
</tr>
<tr>
<td>18</td>
<td>0.49241</td>
<td>0.48740</td>
<td>0.46889</td>
<td>0.43157</td>
<td>0.37249</td>
<td></td>
</tr>
<tr>
<td>19</td>
<td>0.49657</td>
<td>0.47848</td>
<td>0.47219</td>
<td>0.44057</td>
<td>0.38074</td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>0.48128</td>
<td>0.48582</td>
<td>0.45949</td>
<td>0.42725</td>
<td>0.38812</td>
<td></td>
</tr>
</tbody>
</table>

Average: 0.49554  0.48851  0.46576  0.43902  0.38252

Standard Deviation: 0.01188  0.00784  0.00899  0.00925  0.00834

95% Confidence Interval: 0.00556  0.00367  0.00421  0.00433  0.00390
Table 11 - Average Chartered Distance Ratio using Random Dispatch for Double Grid Size Case

<table>
<thead>
<tr>
<th>Trial</th>
<th>Average Chartered Distance Ratio</th>
<th>Number of Taxis</th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>5</td>
<td>10</td>
<td>20</td>
<td>40</td>
<td>80</td>
</tr>
<tr>
<td>1</td>
<td>0.50405</td>
<td>0.53409</td>
<td>0.72623</td>
<td>0.84019</td>
<td>0.90539</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>0.50686</td>
<td>0.53636</td>
<td>0.70410</td>
<td>0.84128</td>
<td>0.89075</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>0.50838</td>
<td>0.53183</td>
<td>0.71004</td>
<td>0.82781</td>
<td>0.89688</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>0.49832</td>
<td>0.54639</td>
<td>0.69053</td>
<td>0.81853</td>
<td>0.88348</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>0.51622</td>
<td>0.52846</td>
<td>0.70356</td>
<td>0.83025</td>
<td>0.88199</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>0.51526</td>
<td>0.52815</td>
<td>0.67122</td>
<td>0.82600</td>
<td>0.89247</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>0.50414</td>
<td>0.52875</td>
<td>0.68011</td>
<td>0.81816</td>
<td>0.88607</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>0.50205</td>
<td>0.54329</td>
<td>0.68563</td>
<td>0.83589</td>
<td>0.88962</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>0.51195</td>
<td>0.51122</td>
<td>0.67692</td>
<td>0.83433</td>
<td>0.88356</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>0.48869</td>
<td>0.51223</td>
<td>0.68487</td>
<td>0.82591</td>
<td>0.88748</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>0.46923</td>
<td>0.53978</td>
<td>0.67010</td>
<td>0.82101</td>
<td>0.88618</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>0.49798</td>
<td>0.53475</td>
<td>0.65819</td>
<td>0.82000</td>
<td>0.88696</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>0.46252</td>
<td>0.53194</td>
<td>0.65968</td>
<td>0.80759</td>
<td>0.88162</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>0.48672</td>
<td>0.52176</td>
<td>0.66705</td>
<td>0.81473</td>
<td>0.88464</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>0.50100</td>
<td>0.51468</td>
<td>0.65623</td>
<td>0.80122</td>
<td>0.87100</td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>0.50211</td>
<td>0.52803</td>
<td>0.66319</td>
<td>0.80629</td>
<td>0.87886</td>
<td></td>
</tr>
<tr>
<td>17</td>
<td>0.50795</td>
<td>0.52309</td>
<td>0.66484</td>
<td>0.78842</td>
<td>0.86838</td>
<td></td>
</tr>
<tr>
<td>18</td>
<td>0.49279</td>
<td>0.54034</td>
<td>0.66113</td>
<td>0.80111</td>
<td>0.85601</td>
<td></td>
</tr>
<tr>
<td>19</td>
<td>0.50237</td>
<td>0.50909</td>
<td>0.63803</td>
<td>0.79012</td>
<td>0.86651</td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>0.50145</td>
<td>0.50611</td>
<td>0.64877</td>
<td>0.77172</td>
<td>0.86123</td>
<td></td>
</tr>
<tr>
<td>Average</td>
<td><strong>0.49900</strong></td>
<td><strong>0.52752</strong></td>
<td><strong>0.67602</strong></td>
<td><strong>0.81603</strong></td>
<td><strong>0.88195</strong></td>
<td></td>
</tr>
<tr>
<td>Standard Deviation</td>
<td>0.01370</td>
<td>0.01181</td>
<td>0.02229</td>
<td>0.01856</td>
<td>0.01209</td>
<td></td>
</tr>
<tr>
<td>95% Confidence Interval</td>
<td>0.00641</td>
<td>0.00553</td>
<td>0.01043</td>
<td>0.00869</td>
<td>0.00566</td>
<td></td>
</tr>
</tbody>
</table>
Table 12 - Average Chartered Distance Ratio using Vehicle Anticipation for Double Grid Size Case

<table>
<thead>
<tr>
<th>Average Chartered Distance Ratio</th>
<th>Number of Taxis</th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>5</td>
<td>10</td>
<td>20</td>
<td>40</td>
<td>80</td>
</tr>
<tr>
<td>1</td>
<td>0.50113</td>
<td>0.45674</td>
<td>0.46064</td>
<td>0.39489</td>
<td>0.33897</td>
</tr>
<tr>
<td>2</td>
<td>0.50470</td>
<td>0.49281</td>
<td>0.44634</td>
<td>0.40946</td>
<td>0.32334</td>
</tr>
<tr>
<td>3</td>
<td>0.47197</td>
<td>0.50012</td>
<td>0.45616</td>
<td>0.39647</td>
<td>0.32948</td>
</tr>
<tr>
<td>4</td>
<td>0.49782</td>
<td>0.48047</td>
<td>0.43127</td>
<td>0.42517</td>
<td>0.31606</td>
</tr>
<tr>
<td>5</td>
<td>0.48522</td>
<td>0.48663</td>
<td>0.44494</td>
<td>0.38736</td>
<td>0.33540</td>
</tr>
<tr>
<td>6</td>
<td>0.50949</td>
<td>0.49150</td>
<td>0.45723</td>
<td>0.39908</td>
<td>0.31840</td>
</tr>
<tr>
<td>7</td>
<td>0.51776</td>
<td>0.47048</td>
<td>0.45103</td>
<td>0.38293</td>
<td>0.28343</td>
</tr>
<tr>
<td>8</td>
<td>0.48993</td>
<td>0.48760</td>
<td>0.42228</td>
<td>0.41412</td>
<td>0.30802</td>
</tr>
<tr>
<td>9</td>
<td>0.47907</td>
<td>0.47400</td>
<td>0.44424</td>
<td>0.40301</td>
<td>0.34499</td>
</tr>
<tr>
<td>10</td>
<td>0.50843</td>
<td>0.49259</td>
<td>0.45992</td>
<td>0.39614</td>
<td>0.32640</td>
</tr>
<tr>
<td>11</td>
<td>0.51131</td>
<td>0.48479</td>
<td>0.44794</td>
<td>0.39750</td>
<td>0.33701</td>
</tr>
<tr>
<td>12</td>
<td>0.48485</td>
<td>0.46482</td>
<td>0.47156</td>
<td>0.42627</td>
<td>0.31682</td>
</tr>
<tr>
<td>13</td>
<td>0.48901</td>
<td>0.49754</td>
<td>0.43175</td>
<td>0.39381</td>
<td>0.31705</td>
</tr>
<tr>
<td>14</td>
<td>0.46638</td>
<td>0.48866</td>
<td>0.45653</td>
<td>0.39890</td>
<td>0.32865</td>
</tr>
<tr>
<td>15</td>
<td>0.47897</td>
<td>0.47797</td>
<td>0.46280</td>
<td>0.38355</td>
<td>0.30758</td>
</tr>
<tr>
<td>16</td>
<td>0.50451</td>
<td>0.48516</td>
<td>0.43909</td>
<td>0.41028</td>
<td>0.27719</td>
</tr>
<tr>
<td>17</td>
<td>0.51597</td>
<td>0.46969</td>
<td>0.45745</td>
<td>0.40177</td>
<td>0.30600</td>
</tr>
<tr>
<td>18</td>
<td>0.49156</td>
<td>0.49993</td>
<td>0.44184</td>
<td>0.40054</td>
<td>0.33777</td>
</tr>
<tr>
<td>19</td>
<td>0.44924</td>
<td>0.48669</td>
<td>0.46828</td>
<td>0.39542</td>
<td>0.29994</td>
</tr>
<tr>
<td>20</td>
<td>0.49144</td>
<td>0.49446</td>
<td>0.43516</td>
<td>0.39657</td>
<td>0.30959</td>
</tr>
<tr>
<td>Average</td>
<td>0.49244</td>
<td>0.48413</td>
<td>0.44932</td>
<td>0.40066</td>
<td>0.31810</td>
</tr>
<tr>
<td>Standard Deviation</td>
<td>0.01770</td>
<td>0.01195</td>
<td>0.01316</td>
<td>0.01164</td>
<td>0.01804</td>
</tr>
<tr>
<td>95% Confidence Interval</td>
<td>0.00828</td>
<td>0.00559</td>
<td>0.00616</td>
<td>0.00545</td>
<td>0.00844</td>
</tr>
</tbody>
</table>
Table 13 – Simulation Parameters for Base Case, Double Frequency Case, and Double Grid Size Case

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Base Case</th>
<th>Double Call Frequency Case</th>
<th>Double Grid Size Case</th>
</tr>
</thead>
<tbody>
<tr>
<td>num_taxi</td>
<td>5, 10, 20, 40, 80</td>
<td></td>
<td></td>
</tr>
<tr>
<td>grid_size_x</td>
<td>32</td>
<td>32</td>
<td>64</td>
</tr>
<tr>
<td>grid_size_y</td>
<td>26</td>
<td>26</td>
<td>52</td>
</tr>
<tr>
<td>call_prob</td>
<td>0.5</td>
<td>1.0</td>
<td>0.5</td>
</tr>
<tr>
<td>end_time</td>
<td>24</td>
<td></td>
<td></td>
</tr>
<tr>
<td>start_time</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>max_time_step</td>
<td>0.3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>incoming_rush_start</td>
<td>7.00</td>
<td></td>
<td></td>
</tr>
<tr>
<td>incoming_rush_end</td>
<td>10.00</td>
<td></td>
<td></td>
</tr>
<tr>
<td>outgoing_rush_start</td>
<td>16.00</td>
<td></td>
<td></td>
</tr>
<tr>
<td>outgoing_rush_end</td>
<td>19.00</td>
<td></td>
<td></td>
</tr>
<tr>
<td>rush_call_increase_factor</td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>rush_call_prob</td>
<td>0.9</td>
<td></td>
<td></td>
</tr>
<tr>
<td>taxi_speed</td>
<td>30</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Appendix A: Source Code

The code for the taxi dispatching environment and the dispatching algorithms is presented here. It is divided into several files (whose names are indicated in the brackets). It is fully commented and can be understood fairly easily.

Main Simulation Module (File: taxi_simulation.py)

#!/usr/bin/env python

# Import Standard Modules
import random  # imports random module (used to generate random numbers)
import math  # imports math module (for mathematical functions)

# Import Modules Created by Author
from class_taxi import Taxi  # imports taxi class
from class_auto_dispatch import Auto_Dispatch  # imports auto_dispatch class
from class_grid import Grid  # imports grid class
from class_call_event_generator import Call_Event_Generator  # imports Call Event Generator Class

# Title
"Simulation on Taxi Dispatching"

# Main Simulation Function
# This is part of the main program which runs the simulation
if __name__ == "__main__":  # If this script is run as a program:

    #------------------------Simulation Parameters
    num_taxi = 5  # number of taxi's
    grid_size_x = 32  # x-size of grid (in km)
    grid_size_y = 26  # y-size of grid (in km)
    end_time = 24  # length of time the simulation must run for (in hours)
    call_prob = 0.5  # probability that an incoming call will occur
    start_time = 0  # starting time of the simulation
    current_time = start_time  # the current time of the simulation
    max_time_step = 0.3  # the maximum step that can occur (resolution)
    time_intvl = 0  # the time interval for the simulation to run.

    #------------------------Rush Hours
    incoming_rush_start = 7.00  # Start Time for Rush our going into Downtown
    incoming_rush_end = 10.00  # End Time for Rush our going into Downtown
    outgoing_rush_start = 16.00  # Start Time for Rush our going out of Downtown
    outgoing_rush_end = 19.00  # End Time for Rush our going out of Downtown

    rush_call_increase_factor = 3  # The number of times calls will increase during rush hour
    rush_call_prob = 0.9  # The probability that the call will be going in the direction of the rush hour traffic

    #------------------------Matrices to Store Results
    taxi_time_loc_status = []  # Stores Taxi Locations and Statuses for each taxi
    for i in range (0, num_taxi):  # at each time Interval
        taxi_time_loc_status.append([])
time_tracker = []  # Keeps track of all time intervals

taxi_speed = 30  # Speed of taxis (in km/hour)
taxi_direction = 0
taxi_location_x = 0
taxi_location_y = 0
taxi_status = 0

# Generates required objects

city = Grid(grid_size_x, grid_size_y, (incoming_rush_start, incoming_rush_end),
(outgoing_rush_start, outgoing_rush_end), rush_call_prob)  # Generates virtual city
city.divide_grid()  # Divides Grid into Sectors
call_evt = Call_Event_Generator(call_prob, city)  # Creates a call event object

# creates the required number of taxi's in a list
taxi = []
for i in range(0, num_taxi):
    # Randomly generates start locations
    taxi_location_x = random.uniform(0, grid_size_x)
taxi_location_y = random.uniform(0, grid_size_y)
taxi.append(Taxi(taxi_location_x, taxi_location_y, taxi_status, taxi_speed, taxi_direction))

dispatcher = Auto_Dispatch(taxi, num_taxi, city, current_time)  # Creates an Auto Dispatch Module

# Starts and runs the simulation

# Simulation will run till end time
while (current_time < end_time):
    call_evt.generate_call(current_time)  # Generate a call event (based on sector and time - rush hour)

    # If there is a call, pass it to the rules object to handle it accordingly
    if (call_evt.call_origin_x >= 0):
        # Adds to RFS queue
        dispatcher.add_RFS_queue(call_evt.call_origin_x,
call_evt.call_origin_y, call_evt.call_destination_x,
call_evt.call_destination_y, current_time)

        # Passes info to dispatcher and gets back changed coords
        # Update Dispatcher with new taxi statuses
        dispatcher.taxi_update(taxi, current_time)

        # Ask Dispatcher to Assign any RFS
        dispatcher.assign_taxis()

    # Gets info back from Dispatcher
    taxi = dispatcher.taxi

    #--------
#Create time interval
#Time interval changes based on rush hours (increases frequency of calls during rush hours)

#Checks whether the system is in rush hour
if ((current_time >= incoming_rush_start) & (current_time <= incoming_rush_end)) | ((current_time >= outgoing_rush_start) & (current_time <= outgoing_rush_end)):
    #If it is in rush hour, interval frequency increases
    time_intvl = random.uniform(0, max_time_step/rush_call_increase_factor) # generates random number 1.0 <= t < max_time_step/rush_call_increase_factor

#Not in rush hour
else:
    time_intvl = random.uniform(0, max_time_step) # generates random number 1.0 <= t < max_time_step

#Checks if time interval will exceed end time.
#If it does, assigns end time to the simulation (i.e. stops at end time)
if (current_time + time_intvl) >= end_time:
    time_intvl = end_time - current_time

#---------Moves Taxis

while (time_intvl > 0):
    #Finds taxi that will be done the soonest and records its time
    least_time_left = time_intvl
    for i in range (0, num_taxi):
        taxi[i].calculate_time_left()
        if (((taxi[i].time_left) < least_time_left) & ((taxi[i].time_left) > 0)):
            least_time_left = taxi[i].time_left

    #Updates current time and remaining time interval
    current_time = current_time + least_time_left
    time_intvl = time_intvl - least_time_left

    #Adds to time_tracker, used for statistics, debugging
    time_tracker.append(current_time)

    #Go through each taxi and move it
    for i in range (0, num_taxi):
        taxi[i].move(least_time_left) #Moves Taxi
        #Updates list of taxis locations and statuses to be used for results
        taxi_time_loc_status[i].append([taxi[i].location_x, taxi[i].location_y, taxi[i].status])

#---------Passes info to dispatcher and gets back changed coords

#Update Dispatcher with new taxi statuses
dispatcher.taxi_update(taxi, current_time)
#Ask Dispatcher to Assign any RFS
dispatcher.assign_taxis()

#Gets info back from Dispatcher
taxi = dispatcher.taxi

#----------------
#---------
#---------

#----------------
#---------
#---------

#Prints Relevant Results
#Go through each taxi and print results
avg_distance_travelled = 0
avg_distance_travelled_call = 0
avg_distance_chartered = 0

total_distance_taxis_travelled = 0
total_distance_taxis_chartered = 0
total_distance_taxis_call = 0

total_time_idle = 0

total_time_chartered = 0

for i in range (0,num_taxi):
    total_distance_taxis_travelled = total_distance_taxis_travelled +
taxi[i].distance_travelled
    total_distance_taxis_chartered = total_distance_taxis_chartered +
taxi[i].distance_chartered
    total_distance_taxis_call = total_distance_taxis_call +
taxi[i].distance_travelled_call
    total_time_chartered = total_time_chartered +
(taxi[i].distance_chartered)/taxi_speed

    total_time_idle = (end_time-
start_time)*num_taxi - total_time_chartered

#Go through answered calls and print results
print "Statistics for Answered Calls:\n"
print "Call \t Location_x \t Location_y \t Target_x \t Target_y \t Call_Time \t Taxi_Assigned \t Time_Answered \t Time_taken_to_Answer\n"

for i in range (0,len(dispatcher.RFS_answered)):
    print "%d \t %f \t %f \t %f \t %f \t %f \t %f \t %d \t %f \t %f" % (i,
dispatcher.RFS_answered[i][0], dispatcher.RFS_answered[i][1],
dispatcher.RFS_answered[i][2], dispatcher.RFS_answered[i][3],
dispatcher.RFS_answered[i][4], dispatcher.RFS_answered[i][5],
dispatcher.RFS_answered[i][6], dispatcher.RFS_answered[i][7])

    total_time_waited = total_time_waited +
dispatcher.RFS_answered[i][7] #Calculates Total waiting time

    average_time_waited = total_time_waited/(len(dispatcher.RFS_answered))

#Prints Parameters of Run:
print "\n",
print dispatcher.dispatch_rules_used
print "Number of Taxis: %i" % (num_taxi)
print "Call Probability at any given interval: %f" % (call_prob)
print "Grid Size X: %f" % (grid_size_x)
print "Grid Size Y: %f" % (grid_size_y)
print "Simulation Run Time (Hours): %f" % (end_time-start_time)
print "Taxi Speed: %f" % (taxi_speed)

# Call Stats
print "Average Call Waiting Time: %f" % (average_time_waited)
print "Average Utilisation (distance chartered/distance travelled): %f" % (total_distance_taxis_chartered/total_distance_taxis_travelled)
print "Total Number of Calls: %d" % (len(dispatcher.RFS_queue) + len(dispatcher.RFS_answered))
print "Total Number of Calls Answered: %d" % len(dispatcher.RFS_answered)
print "Total Number of Calls Unanswered: %d" % len(dispatcher.RFS_queue)

# Utilisation Stats
print "Average Distance Travelled: %f" % (total_distance_taxis_travelled/num_taxi)
print "Average Distance Chartered: %f" % (total_distance_taxis_chartered/num_taxi)
print "Average Distance Travelled to a Call: %f" % (total_distance_taxis_call/num_taxi)
print "Average Distance Travelled Idle: %f" % ((total_distance_taxis_travelled-total_distance_taxis_chartered)/num_taxi)

# Print Data in Column Format:
print "Average Call Waiting Time, Average Utilisation, Total Number of Calls, Total Number of Calls Answered, Total Number of Calls Unanswered, Average Distance Travelled, Average Distance Chartered, Average Distance Travelled to a Call, Average Distance Travelled Idle,"
print "%f, %f, %d, %d, %d, %f, %f, %f," % (average_time_waited, (total_distance_taxis_chartered/total_distance_taxis_travelled), (len(dispatcher.RFS_queue) + len(dispatcher.RFS_answered)), len(dispatcher.RFS_answered), len(dispatcher.RFS_queue), (total_distance_taxis_travelled/num_taxi), (total_distance_taxis_chartered/num_taxi), (total_distance_taxis_call/num_taxi), ((total_distance_taxis_travelled-total_distance_taxis_chartered)/num_taxi))

Taxi Class (File: class_taxi.py)

# Import Standard Modules
import random # imports random module (used to generate random numbers)
import math # imports math module (for mathematical functions)

# Taxi Class
# This class represents an individual taxi
class Taxi:

    # Class Constructor (Initialisation)
def __init__(self, vehicle_location_x, vehicle_location_y, vehicle_status, vehicle_speed, vehicle_direction):
        self.location_x = vehicle_location_x # initial x-coord location
        self.location_y = vehicle_location_y # initial y-coord location
        self.status = vehicle_status # initial status (0 = not chartered, 1 = chartered, 2 = Going to a Call Location, 3 = Picked up Customer/Reached RFS origin and Waiting for final target)
        self.speed = vehicle_speed # initial speed (in grid units / unit time)
        self.direction = vehicle_direction # in radians, 0 rads is due north, positive is clockwise
        self.distance_chartered = 0 # total distance chartered (in grid units)
self.distance_travelled = 0 # total distance travelled (in grid units)
self.distance_travelled = 0 # total distance travelled to get to a call (has a status of 2)

self.time_left = 0 # the time that remains for current job

# Monitor Variables

# Changes vehicles status

# Changes vehicles direction

# Changes vehicles target

# Calculate distance between two points

def calculate_distance(x1, y1, x2, y2):
    distance = math.sqrt((x2 - x1)*(x2 - x1) + (y2 - y1)*(y2 - y1))

# Calculates Time left till job is finished

def calculate_time_left(self):
    # If the taxi is not staying idle
    if ((self.target_x > -1) & (self.target_y > -1)):
        distance = self.calculate_distance(self.location_x,
                                            self.location_y, self.target_x, self.target_y) # Gets distance to target
        self.time_left = distance / self.speed
    else:
        self.time_left = 0

# Moves vehicle in current direction at current speed and changes location at end of move

def move(self, time_intvl):
    if ((self.target_x > -1) & (self.target_y > -1)):
        self.calculate_direction() # Calculates direction the vehicle is travelling in (used for calculations)
        self.calculate_time_left() # calculates time till it reaches target

        # Checks if the vehicle will reach its target before the time interval is up (at end of move, every taxi will be in this case)
        if (time_intvl >= self.time_left):
            self.target_x = -1
            self.target_y = -1

            # Changes vehicles status
            self.status = vehicle_status

            # Changes vehicles direction
            self.direction = vehicle_direction

            # Changes vehicles target
            self.target_x = vehicle_target_x
            self.target_y = vehicle_target_y

            # Calculates direction the vehicle is travelling in (used for calculations)
            self.calculate_direction()

            # Calculates time till it reaches target

            # Calculates distance between two points
            return distance

# Changes vehicles status

def change_status(self, vehicle_status):
    self.status = vehicle_status

# Changes vehicles direction

def change_direction(self, vehicle_direction):
    self.direction = vehicle_direction

# Changes vehicles target

def change_target(self, vehicle_target_x, vehicle_target_y):
    self.target_x = vehicle_target_x
    self.target_y = vehicle_target_y

    self.calculate_direction() # Calculates direction the vehicle is travelling in (used for calculations)
    self.calculate_time_left() # calculates time till it reaches target

# Calculates distance between two points

def calculate_distance(x1, y1, x2, y2):
    distance = math.sqrt((x2 - x1)*(x2 - x1) + (y2 - y1)*(y2 - y1))

# Calculates Time left till job is finished

def calculate_time_left(self):
    # If the taxi is not staying idle
    if ((self.target_x > -1) & (self.target_y > -1)):
        distance = self.calculate_distance(self.location_x,
                                            self.location_y, self.target_x, self.target_y) # Gets distance to target
        self.time_left = distance / self.speed
    else:
        self.time_left = 0

# Moves vehicle in current direction at current speed and changes location at end of move

def move(self, time_intvl):
    # Only move if the target coordinates are >-1
    if ((self.target_x > -1) & (self.target_y > -1)):
        self.calculate_direction() # Calculates direction the vehicle is travelling in (used for calculations)
        self.calculate_time_left() # calculates time till it reaches target

        # Checks if the vehicle will reach its target before the time interval is up (at end of move, every taxi will be in this case)
        if (time_intvl >= self.time_left):
self.time_left = 0 # Time interval is done

# Calculates distance travelled
distance_moved = self.calculate_distance(self.location_x, self.location_y, self.target_x, self.target_y)

# Change location to target location
self.location_x = self.target_x
self.location_y = self.target_y

# Resets target location
self.change_target(-1, -1)

# If reached call origin, change status and waits for destination coords
if (self.status == 2):
    self.status = 3
    self.distance_travelled_call = self.distance_travelled_call + distance_moved

    # If completed call, change status, wait for next move from dispatcher
    if (self.status == 1):
        self.status = 0
        self.distance_chartered = self.distance_chartered + distance_moved

# Otherwise if move is not done in time interval
else:

    # Calculate component speeds
    x_speed = self.speed * math.cos(self.direction)
    y_speed = self.speed * math.sin(self.direction)

    # Calculate new coordinates and distance travelled
    new_location_x = self.location_x + x_speed * time_intvl
    new_location_y = self.location_y + y_speed * time_intvl
    distance_moved = self.calculate_distance(self.location_x, self.location_y, new_location_x, new_location_y)

    # Move taxi to location travelled in time interval
    self.location_x = new_location_x
    self.location_y = new_location_y

    self.time_left = 0 # Indicated no time left in current interval

    # Updates distance moved to a call
    if (self.status == 2):
        self.distance_travelled_call = self.distance_travelled_call + distance_moved

    # Updates chartered distance
    if (self.status == 1):
        self.distance_chartered = self.distance_chartered + distance_moved

    # Update total distance travelled
    self.distance_travelled = self.distance_travelled + distance_moved

    # Append to list of taxi locations (for results)
    self.result_taxi_location_status.append([self.location_x,
# Calculate angle (direction) of the vehicle

def calculate_direction(self):
    # Check non-zero division:
    if ((self.target_x - self.location_x)==0):
        # If going straight up
        if (self.target_y > self.location_y):
            self.direction = math.pi/2
        # If going straight down
        elif (self.target_y < self.location_y):
            self.direction = 3*math.pi/2

    # Check unbounded error
    elif ((self.target_y - self.location_y)==0):
        # If going straight right
        if (self.target_x > self.location_x):
            self.direction = 0
        # If going straight left
        elif (self.target_x < self.location_x):
            self.direction = math.pi

    # If not non-zero division
    else:
        # Calculate slope of line and proceed
        slope = (self.target_y - self.location_y) / (self.target_x - self.location_x)

        # Case quadrant I:
        if ((self.target_x >= self.location_x) and (self.target_y >= self.location_y)):
            self.direction = math.atan(slope)
        # Case quadrant II:
        elif ((self.target_x < self.location_x) and (self.target_y > self.location_y)):
            self.direction = math.atan(slope) + math.pi
        # Case quadrant III:
        elif ((self.target_x < self.location_x) and (self.target_y < self.location_y)):
            self.direction = math.atan(slope) + math.pi
        # Case quadrant IV:
        elif ((self.target_x >= self.location_x) and (self.target_y < self.location_y)):
            self.direction = math.atan(slope) + 2*math.pi

Grid Class (File: class_grid.py)

# Import Standard Modules
import random  # imports random module (used to generate random numbers)
import math  # imports math module (for mathematical functions)

# Grid Class
# This class represents the city as a grid
# Assumes the grid is rectangular
class Grid:

    # Class Constructor (Initialisation)
    def __init__(self, grid_size_x, grid_size_y, in_rush_hour, out_rush_hour, rush_hour_call_prob):
        self.size_x = grid_size_x  # x length of rectangular city
        self.size_y = grid_size_y  # y length of rectangular city

        self.divide_grid()  # Divides Grid into sectors

        self.incoming_rush_start = in_rush_hour[0]  # Start Time for Rush our going into Downtown
        self.incoming_rush_end = in_rush_hour[1]  # End Time for Rush our going into Downtown

        self.outgoing_rush_start = out_rush_hour[0]  # Start Time for Rush our going out of Downtown
        self.outgoing_rush_end = out_rush_hour[1]  # End Time for Rush our going out of Downtown

        self.rush_call_prob = rush_hour_call_prob  # The probability that the call will be going in the direction of the rush hour traffic

    # Changes the size of the grid
    def change_grid_size(self, grid_size_x, grid_size_y):
        self.size_x = grid_size_x
        self.size_y = grid_size_y

        self.divide_grid(self)  # Divides Grid into Sectors

        # Sectors must be rectangular in shape
        # Initial coordinate is left bottom, final is right top
        # Sector list must be in decreasing order of demand, with downtown first
        def divide_grid(self):
            # Initialise Sectors
            self.sectors = []  # Sectors Structures as: [(0): demand (as a decimal - it is the probability that this location will be the pickup/drop off location), (1): initial coords(x,y), (2): final coords(x,y)]

            # Sector 0 - Downtown
            self.sectors.append([0.4, ((0.25) * (self.size_x), 0), ((0.75) * (self.size_x), (0.5) * (self.size_y))])

            # Sector 1 - West
            self.sectors.append([0.2, (0, 0), ((0.25) * (self.size_x), self.size_y)])

            # Sector 2 - North
            self.sectors.append([0.2, ((0.25) * (self.size_x), (0.5) * (self.size_y)), ((0.75) * (self.size_x), (self.size_y))])

            # Sector 3 - East
            self.sectors.append([0.2, ((0.75) * (self.size_x), 0), (self.size_x, self.size_y)])

            # Sorts sectors in descending order of demand
            self.sectors.sort(lambda x, y: cmp(y[0], x[0]))

            # Function Checks which sector a given coordinate is in and returns the
sector number (i.e. index in sector list)
#Returns -1 if sector not found
#Returns FIRST sector if found
def find_sector(self, location_x, location_y):
    which_sector = -1 #The index of the sector the location is in
    #cycle through sectors
    for i in range (0, len(self.sectors)):
        #Compare Coordinates to see it it is in the sector
        if ((location_x >= self.sectors[i][1][0])&(location_y >=
        self.sectors[i][1][1])&(location_x <= self.sectors[i][2][0])&(location_y <=
        self.sectors[i][2][1])):
            which_sector = i
            break
    return i

#Function Generates Random Coordinates within a given sector
def random_sector_location(self, which_sector):
    #Generates Coordinates in the given sector
    destination_x = random.uniform(self.sectors[which_sector][1][0],
    self.sectors[which_sector][2][0]) # x coord of random location in sector
    destination_y = random.uniform(self.sectors[which_sector][1][1],
    self.sectors[which_sector][2][1]) # y coord of random location in sector
    return (destination_x, destination_y)

#Function Decides which sector will be selected based on the demand
#probabilities in a sector
#It does this using a weighted random distribution
#It Basically randomly selects a sector based on the sector weights
def weighted_sector_selection(self, rush_hour_flag = None):
    #the rush_hour_flag indicates whether to take downtown into
    #consideration
    #if no flag is present, then downtown isnt taken into consideration
    #if it is set (as 1 or any value), then downtown is NOT taken into
    #account
    if rush_hour_flag is None:
        start_sector = 0
    else:
        start_sector = 1
    #Gets total of demand weights
    weight_sum = 0
    for i in range(start_sector, len(self.sectors)):
        weight_sum = weight_sum + self.sectors[i][0]
    #Generates random number between 0 and weightsum
    rnd = random.uniform(0, weight_sum)
    #Goes through and sees which interval the random value lies
    #The interval of the weight it lies in is the sector
    for i in range(start_sector, len(self.sectors)):
        if (rnd < self.sectors[i][0]):
            weighted_random_sector = i
            break
        else:
            rnd = rnd - self.sectors[i][0]
return weighted_random_sector

Call Event Generator Class (File: class_call_event_generator)

# Import Standard Modules
import random # imports random module (used to generate random numbers)
import math # imports math module (for mathematical functions)

# Call Event Generator Class
# This class generates call events in the city.
class Call_Event_Generator:
    # Class Constructor (Initialisation)
    def __init__(self, call_probability, city_object):
        self.call_prob = call_probability # probability the taxi will get a request for service call
        self.city = city_object # The grid used
        self.size_x = city_object.size_x # x length of rectangular city
        self.size_y = city_object.size_y # y length of rectangular city
        self.reset_coords()

    # Resets origin and destination coords.
    def reset_coords(self):
        self.call_origin_x = -1 # x coord of call location (origin)
        self.call_origin_y = -1 # y coord of call location (origin)
        self.call_destination_x = -1 # x coord of call target (destination)
        self.call_destination_y = -1 # y coord of call target (destination)

    # Function Generates the required call
    def generate_call(self, current_time):
        self.reset_coords() # Resets any previous Coords.
        rnd_call = random.random() # creates a random variable to check probability of an incoming call

        # There is an incoming call if the random call variable is less than the call porbability
        if rnd_call < self.call_prob:
            # Checks whether the call is in rush hour
            if ((current_time >= self.city.incoming_rush_start)&(current_time <= self.city.incoming_rush_end))|((current_time >= self.city.outgoing_rush_start)&(current_time <= self.city.outgoing_rush_end)):
                # Checks whether the call is in the rush hour direction
                rnd_call = random.random() # creates a random variable to check call direction
                if rnd_call < self.city.rush_call_prob: # If it is in rush hour
                    # CALL INCOMING RUSH GENERATION FUNCTION to generate call from suburb into downtown
self.generate_rush_call_in()

elif ((current_time >=
self.city.outgoing_rush_start)&(current_time <= self.city.outgoing_rush_end)):
  #If outgoing rush
  #CALL OUTGOING RUSH GENERATION FUNCTION to generate
call from downtown to suburb
  self.generate_rush_call_out()

else:
  #Call not in rush hour direction
  #GENERATE NORMAL SECTOR CALL
  self.generate_sector_call()

else:
  #Call not in rush hour
  #GENERATE NORMAL SECTOR CALL
  self.generate_sector_call()

# Finds location of call and its destination
# If coordinates are negative, then there is no call event
# Note that this function Generates calls according to the sectors given
def generate_sector_call(self):

  #Selects the origin sector of the call given the weights (i.e. randomly
selects a sector given weighting of demands)
  origin_sector = self.city.weighted_sector_selection()

  #sets random coords. in origin sector as call origin location
  #All pickup destinations are within the sector limits
  (self.call_origin_x, self.call_origin_y) =
  self.city.random_sector_location(origin_sector)

  #Decides in which sector call destination is (Based on demand
probability of the sectors)
  destination_sector = self.city.weighted_sector_selection()

  #sets random coords. in destination sector as pickup destination
  #All pickup destinations are within the sector limits
  (self.call_destination_x, self.call_destination_y) =
  self.city.random_sector_location(destination_sector)

# Generates Origin and Destination coords for a call going INTO Downtown
def generate_rush_call_in(self):

  #Selects the origin sector of the call given the weights (i.e. randomly
selects a sector given weighting of demands)
  #Note, the origin is only selected from the suburbs
  origin_sector = self.city.weighted_sector_selection(1)

  #sets random coords. in origin sector as call origin location
  #All pickup destinations are within the sector limits
  (self.call_origin_x, self.call_origin_y) =
  self.city.random_sector_location(origin_sector)

  #Sets Destination sector of call as downtown
  destination_sector = 0

  #sets random coords. in destination sector as pickup destination
  #All pickup destinations are within the sector limits
  (self.call_destination_x, self.call_destination_y) =
  self.city.random_sector_location(destination_sector)
# Generates Origin and Destination coords for a call going OUT OF Downtown

def generate_rush_call_out(self):
    # Selects the origin sector of the call as downtown
    origin_sector = 0

    # sets random coords. in origin sector as call origin location
    # All pickup destinations are within the sector limits
    (self.call_origin_x, self.call_origin_y) =
    self.city.random_sector_location(origin_sector)

    # Sets Destination sector of call as a suburb
    # Note, the destination is only selected from the suburbs
    destination_sector = self.city.weighted_sector_selection(1)

    # sets random coords. in destination sector as pickup destination
    # All pickup destinations are within the sector limits
    (self.call_destination_x, self.call_destination_y) =
    self.city.random_sector_location(destination_sector)

    # Checks whether there is a call and gives location (i.e. generates random call)
    # The call location and destination are completely random
    def generate_random_call(self):
        self.reset_coords()  # Resets any previous Coords.

        rnd_call = random.random()  # creates a random variable to check probability of an incoming call

        # There is an incoming call if the random call variable is less than the call porbability
        if rnd_call < self.call_prob:
            # sets random coords. as call origin and destination
            # All call origins/destinations are within the grid limits
            self.call_origin_x = random.uniform(0, self.size_x)  # x coord of call location (origin)
            self.call_origin_y = random.uniform(0, self.size_y)  # y coord of call location (origin)

            self.call_destination_x = random.uniform(0, self.size_x)  # x coord of call target (destination)
            self.call_destination_y = random.uniform(0, self.size_y)  # y coord of call target (destination)

Auto Dispatch Class (File: class_auto_dispatch.py)

# Import Standard Modules
import random  # imports random module (used to generate random numbers)
import math  # imports math module (for mathematical functions)

# Automated Dispatching Class
# Decides how taxis should react when idle or when they must respond to requests for service

class Auto_Dispatch:
    # Class Constructor (Initialisation)
    def __init__(self, taxi_object, num_taxi, city_object, current_time):
        self.num_taxi = num_taxi  # Number of Taxis
self.current_time = current_time  # Current Time

self.RFS_queue = []  # Initialises queue
# Queue for RFS has following information ((0) call location x, (1) call
# location y, (2) target x, (3) target y, (4) call time, (5) taxi index assigned or
# not = -1))

self.RFS_answered = []  # Initialises queue to hold all answered calls
# Queue for RFS answered has following information ((0) call location x,
# (1) call location y, (2) target x, (3) target y, (4) call time, (5) taxi index
# assigned to it, (6) answer time, (7) time taken to answer)

# Initialise Taxi Information
self.taxi = taxi_object

# Initialise Grid Information
self.city = city_object

# Initialise statistics
self.total_RFS_wait_time = 0

# Indicates which dispatch algorithm was used (for statistics purposes)
self.dispatch_rules_used = ""

# Asks dispatcher to give taxis orders
def assign_taxis(self):
    # The function which contains desired dispatch algorithm is called here.
    # self.auto_dispatch_1() # Uses auto_dispatch_1 (Random Dispatch)
    algorithm to assign taxis
    self.auto_dispatch_3() # Uses auto_dispatch_3 (Vehicle Anticipation)
    algorithm to assign taxis

    # Dispatch Algorithm 1
    # Call Assignment: Assigns closest taxi to given call. Calls answered in
    # order based on time they came in.
    # Idle Movement: Gives idle Taxis random coordinates anywhere in the grid
to move
    def auto_dispatch_1(self):
        # Indicates Which Dispatch Algorithm was Used
        self.dispatch_rules_used = "Dispatch 1 (Random Dispatch)"

        # Assigns closest taxi to given call
        self.assign_RFS_closest_taxi()

        # Ask Dispatcher to Assign final coordinates for answered calls
        self.assign_answered()

        # Ask Dispatcher to Assign Give idle Movement Orders
        # If no Idle Movements, then the taxis will remain stationary at their
        last point

    # Dispatch Algorithm 2
    # Call Assignment: Assigns closest taxi to given call. Calls answered in
    # order based on time they came in.
    # Idle Movement: Tried to keep atleast 1 taxi in every sector. Number of
taxis in a given sector is
    # proportional to the demand weighting in that sector
    def auto_dispatch_2(self):
# Indicates Which Dispatch Algorithm was Used
self.dispatch_rules_used = "Dispatch 2 (Sector Dependent Dispatch)"

# Assigns closest taxi to given call
self.assign_RFS_closest_taxi()

# Ask Dispatcher to Assign final coordinates for answered calls
self.assign_answered()

# Ask Dispatcher to Assign Give idle Movement Orders
# Gives idle Taxis coordinates to distribute them in the grid
self.assign_idle_sector_random_coords()

# Dispatch Algorithm 3
# Call Assignment: Assigns closest taxi to given call. Calls answered in order based on time they came in.
# Idle Movement: Tries to keep atleast 1 taxi in every sector. Number of taxis in a given sector is proportional to the demand weighting in that sector. It sends taxis into the centre of the sectors, where they are idle till called. This module also takes into account rush hours.
def auto_dispatch_3(self):
    # Indicates Which Dispatch Algorithm was Used
    self.dispatch_rules_used = "Dispatch 3 (Vehicle Anticipation)"

    # Assigns closest taxi to given call
    self.assign_RFS_closest_taxi()

    # Ask Dispatcher to Assign final coordinates for answered calls
    self.assign_answered()

    # Ask Dispatcher to Assign Give idle Movement Orders
    # Gives idle Taxis coordinates to distribute them in the grid
    self.assign_idle_sector_rush_coords()

    # Gets numbers of taxis needed in given sectors based on time of day (takes rush hour into account)
def get_needed_taxis(self):
    needed_taxis = [0]*(len(self.city.sectors))  # Creates a blank array to store number of taxis needed in each sector
    sector_demand = [0]*(len(self.city.sectors))  # Creates a blank array to store demand in each sector
    lead_time = 0.5  # Lead time is the amount of time (in hours) before to start preparing for rush hour

    # Checks whether the system is in rush hour (or close to it)
    # Checks for incoming rush
    if ((self.current_time) >= (self.city.incoming_rush_start - lead_time)) & (self.current_time) <= (self.city.incoming_rush_end)):
        # Downtown sector demand for taxis is the rush hour call probability
        sector_demand[0] = (1-self.city.rush_call_prob)

    # Checks for outgoing rush hour
    elif ((self.current_time) >= (self.city.outgoing_rush_start - lead_time)) & (self.current_time) <= (self.city.outgoing_rush_end)):
# Downtown sector demand for taxis is the rush hour call probability
sector_demand[0] = self.city.rush_call_prob

# If neither, then uses downtown demand probability
else:
    sector_demand[0] = self.city.sectors[0][0]

# Gets total of demand weights for rest of sectors
weight_sum = 0
for i in range(1, len(self.city.sectors)):
    weight_sum = weight_sum + self.city.sectors[i][0]

# Call probability for other sectors is normalised with rest of demand, based on individual sector demand
for i in range(1, len(self.city.sectors)):
    sector_demand[i] = (1 - sector_demand[0]) * (self.city.sectors[i][0]) / (weight_sum)

for i in range(0, len(self.city.sectors)):
    # Checks how many taxis needed in the sector (proportional to demand in sector)
    needed_taxis[i] = math.floor((sector_demand[i]) * self.num_taxi)

return needed_taxis

# Assigns sectors for the taxis to go to
# Tries to keep atleast minimum number of taxis in each sector (1st priority)
# If more than more than minimum available, keeps the taxis in the sector dependent on the weighting of demand in the sector
# Takes into account rush hour patterns

def assign_idle_sector_rush_coords(self):
    sector_taxis = [0] * (len(self.city.sectors))  # Creates a blank array to store number of taxis currently in each sector
    assigned_taxis = [0] * self.num_taxi  # Creates a blank array to store which taxis have already been assigned (taxis not assigned = 0, assigned = 1)
    # needed_taxis = [0] * (len(self.city.sectors))  # Creates a blank array to store number of taxis needed in each sector

    # Gets required taxis in each sector
    needed_taxis = self.get_needed_taxis()

    # Checks to see what taxis are in each sector by cycling through them
    # Updates the number of taxis needed in each sector
    for i in range(0, self.num_taxi):
        # Updates list of assigned taxis if they are occupied
        if (self.taxi[i].status != 0):
            assigned_taxis[i] = 1

    # Loops till all taxis assigned or all needs satisfied
    while ((needed_taxis > ([0] * (len(self.city.sectors)))) & (assigned_taxis < ([1] * self.num_taxi))):

        # Cycles through sectors and assigns closest taxi to go to sectors (remember sector list in decreasing order of demand)
        for i in range(0, len(self.city.sectors)):
# If there is a needed taxi in the given sector, then send
# closest idle taxi to the sector centre to the sector centre
if (needed_taxis[i] > 0):
    sector_center_x = (self.city.sectors[i][1][0] +
                      self.city.sectors[i][2][0]) / 2  # X coordinate of center of sector
    sector_center_y = (self.city.sectors[i][1][1] +
                      self.city.sectors[i][2][1]) / 2  # Y coordinate of center of sector

    # Finds closest idle taxi to sector centre
    closest_taxi = self.find_closest_taxi(sector_center_x, sector_center_y, assigned_taxis)

    # If a taxi is available
    if (closest_taxi > -1):
        # Assigns sector center coords to taxi
        self.taxi[closest_taxi].change_target(sector_center_x, sector_center_y)

        # Update list of assigned taxis
        assigned_taxis[closest_taxi] = 1

        # Reduces needed taxis in sector by one
        needed_taxis[i] = needed_taxis[i] - 1

    # Assigns all not assigned taxis to nearest sector center
    # Note: May Not need this with new subtraction formula
    for i in range (0, self.num_taxi):
        if (assigned_taxis[i] == 0):  # If a taxis hasn't been assigned yet
            current_sector = self.city.find_sector(self.taxi[i].location_x, self.taxi[i].location_y)

            # Assign random coords in sector to the taxi
            sector_center_x = (self.city.sectors[current_sector][1][0] +
                                self.city.sectors[current_sector][2][0]) / 2  # X coordinate of center of sector
            sector_center_y = (self.city.sectors[current_sector][1][1] +
                                self.city.sectors[current_sector][2][1]) / 2  # Y coordinate of center of sector

            self.taxi[i].change_target(sector_center_x, sector_center_y)

            # Update assigned taxis
            assigned_taxis[i] = 1

# Asks Dispatcher to Assign final coordinates for answered calls
# Removes RFS from queue and gets statistics
def assign_answered(self):

    # Check each taxi and see if it free
    for i in range (0, self.num_taxi):
        if (self.taxi[i].status == 3):  # Check that taxi has reached the call

            # Searches through and finds the RFS that was answered
            for j in range (0, len(self.RFS_queue)):
                if (self.RFS_queue[j][5] == i):
                    # Updates Taxis coords and status
                    self.taxi[i].change_target(self.RFS_queue[j][2], self.RFS_queue[j][3])

                    self.taxi[i].change_status(1)
#adds call to answered queue FOR STATISTICS
RFS_stat_temp = self.RFS_queue[j]  #Temporarily holds
  
  #Temporarily holds RFS variables so we can manipulate
time_taken = self.current_time - self.RFS_queue[j][4]  #
calculates time taken
RFS_stat_temp.extend([self.current_time, time_taken])

#Extends time answered, time taken to answer to list
self.RFS_answered.append(RFS_stat_temp)  #adds to
answered queue
  
  #Removes given call from queue
self.RFS_queue.pop(j)
break

# Assigns random coords in the whole grid for idle taxis to go to

def assign_idle_random_coords(self):

    #Check each taxi and see if it free
    for i in range (0, self.num_taxi):

        if (self.taxi[i].status == 0):  #Check that taxi is not chartered
            random_coords = self.generate_random_coords(self.city.size_x,
self.city.size_y)  # Generate a random coords for it to go to
            #Changes the target coordinates of the given taxi
            self.taxi[i].change_target(random_coords[0], random_coords[1])

    # Assigns random coords in the given sector for a given taxi to go to
    # Tries to keep atleast minimum number of taxis in each sector (1st priority)
    # If more than more than minimum available, keeps the taxis in the sector
    # dependent on the weighting of demand in the sector

def assign_idle_sector_random_coords(self):

    sector_taxis = [0]*(len(self.city.sectors))  #Creates a blank array to
store number of taxis currently in each sector
assigned_taxis = [0]*self.num_taxi  #Creates a blank array to store
which taxis have already been assigned (taxis not assigned = 0, assigned = 1)
needed_taxis = [0]*(len(self.city.sectors))  #Creates a blank array to
store number of taxis needed in each sector

    #Checks to see what taxis are in each sector by cycling through them
    for i in range (0, self.num_taxi):
        sector = self.city.find_sector(self.taxi[i].location_x,
self.taxi[i].location_y)  #Check which sector that taxi is in
        sector_taxis[sector] = sector_taxis[sector] + 1  #Update the number
of taxis in the given sector
        assigned_taxis[i] = 1  #Updates list of assigned taxis if they are occupied
        if (self.taxi[i].status != 0):
            assigned_taxis[i] = 1

    #Finds required taxis in each sector based on numbers of taxis and
relative sector demand
    for i in range (0, len(self.city.sectors)):

        #Checks how many taxis needed in the sector (proportional to demand
in sector)
        need = math.floor((self.city.sectors[i][0])*self.num_taxi)
        #Updates taxis needed (note, we subtract number of taxis currently
in the sector)
    needed_taxis[i] = need #- sector_taxis[i]

  #If there is no taxi in the given sector, then send closest idle taxi to the sector centre to the sector
  #No need to check if the taxi assigned is out of the sector as there are no taxis in the sector
  if (sector_taxis[i] < 1):
      sector_center_x = (self.city.sectors[i][1][0] +
                        self.city.sectors[i][2][0])/2 # X coordinate of center of sector
      sector_center_y = (self.city.sectors[i][1][1] +
                        self.city.sectors[i][2][1])/2 # Y coordinate of center of sector

      #Finds closest idle taxi to sector centre
      closest_taxi = self.find_closest_taxi(sector_center_x,
                                             sector_center_y, assigned_taxis)

      #If a taxi is available
      if (closest_taxi > -1):
          #Assign random coords in sector to the taxi
          random_sector_coords = self.city.random_sector_location(i)

          self.taxi[closest_taxi].change_target(random_sector_coords[0],
                                                 random_sector_coords[1])

          #Update list of assigned taxis
          assigned_taxis[closest_taxi] = 1

          #reduces needed taxis in sector by one
          needed_taxis[i] = needed_taxis[i] - 1

  #Loops till all taxis assigned or all needs satisfied
  while ((needed_taxis > ([0] * (len(self.city.sectors)))) &
         (assigned_taxis < ([1] * self.num_taxi))):

      #Cycles through sectors and assigns closest taxis to go to sectors (remember sector list in decreasing order of demand)
      for i in range (0, len(self.city.sectors)):

          #If there is a needed taxi in the given sector, then send closest idle taxi to the sector centre to the sector
          if (needed_taxis[i] > 0):
              sector_center_x = (self.city.sectors[i][1][0] +
                                  self.city.sectors[i][2][0])/2 # X coordinate of center of sector
              sector_center_y = (self.city.sectors[i][1][1] +
                                  self.city.sectors[i][2][1])/2 # Y coordinate of center of sector

              #Finds closest idle taxi to sector centre
              closest_taxi = self.find_closest_taxi(sector_center_x,
                                                     sector_center_y, assigned_taxis)

              #If a taxi is available
              if (closest_taxi > -1):
                  #Assign random coords in sector to the taxi
                  random_sector_coords =
                                  self.city.random_sector_location(i)

                  self.taxi[closest_taxi].change_target(random_sector_coords[0],
                                                          random_sector_coords[1])
random_sector_coords[1])

    # Update list of assigned taxis
    assigned_taxis[closest_taxi] = 1

    # Reduces needed taxis in sector by one
    needed_taxis[i] = needed_taxis[i] - 1

    # Assigns all not assigned taxis to random locations within their sectors
    # Note: May Not need this with new subtraction formula
    for i in range (0, self.num_taxi):
        if (assigned_taxis[i] == 0):  # If a taxi hasn't been assigned yet
            print "got here"
            # Check which sector that taxi is in
            current_sector = self.city.find_sector(self.taxi[i].location_x,
                                                    self.taxi[i].location_y)

            # Assign random coords in sector to the taxi
            random_sector_coords =
                self.city.random_sector_location(current_sector)
            self.taxi[i].change_target(random_sector_coords[0],
                                        random_sector_coords[1])

            # Update assigned taxis
            assigned_taxis[i] = 1

    # Adds call to queue. Queue is ordered by time of RFS
    def add_RFS_queue (self, call_loc_x, call_loc_y, target_x, target_y,
                        call_time):
        # Adds call to queue
        self.RFS_queue.append([call_loc_x, call_loc_y, target_x, target_y,
                                call_time, -1])

        # Updates Taxi Info
        def taxi_update (self, taxi_object, current_time):
            self.taxi = taxi_object  # Update taxi objects
            self.current_time = current_time  # Update current time

        # Calculate distance between two points
        def calculate_distance (self, x1, y1, x2, y2):
            distance = math.sqrt((x2 - x1)*(x2 - x1) + (y2 - y1)*(y2 - y1))  # Calculates distance between two points
            return distance

        # Generates idles taxi movements
        # Currently gives it random coordinates to go to
        def generate_random_coords (self, size_x, size_y):
            # Generate random x and y coordinates for the taxi to go to and return
            rand_x = random.uniform(0, size_x)  # x coord
            rand_y = random.uniform(0, size_y)  # y coord
            return (rand_x, rand_y)

        # Assigns Appropriate Taxi to incoming call
        # This currently checks closest available cab to the call and assigns it
        # The call in question is the call index (position in queue starting with 0)
# Returns a list of taxis that have been assigned
def assign_RFS_closest_taxi(self):
    # Cycle through calls
    for i in range(0, len(self.RFS_queue)):
        # If call is un attended
        if (self.RFS_queue[i][5] == -1):
            # Find closest free available taxi to call
            closest_taxi = self.find_closest_taxi(self.RFS_queue[i][0], self.RFS_queue[i][1])
            # Assign taxi to that call if there is one available
            if (closest_taxi > -1):
                self.RFS_queue[i][5] = closest_taxi
            # Change Parameters of the taxi so the target is the call location and that status is going to a pickup
            self.taxi[closest_taxi].change_target(self.RFS_queue[i][0], self.RFS_queue[i][1])
            self.taxi[closest_taxi].change_status(2)

# Checks the Closest idle Taxi to a given location
def find_closest_taxi(self, call_location_x, call_location_y, assigned_taxis = None):
    # Checks if optional assigned_taxis has been called
    if assigned_taxis is None:
        assigned_taxis = [0]*self.num_taxi # If none, assigns a blank array to it
    best_taxi = -1 # List index of best taxi to take the call
    best_distance = -1 # Distance of closest taxi to call

    # Cycle through all taxis
    for i in range(0, self.num_taxi):
        # Check if taxi is free and not otherwise assigned
        if ((self.taxi[i].status == 0) & (assigned_taxis[i] == 0)):
            # Check distance to call
            distance = self.calculate_distance(self.taxi[i].location_x, self.taxi[i].location_y, call_location_x, call_location_y)
            # Make Best Distance if first taxi to be free and assign index to best taxi
            if (best_distance == -1):
                best_distance = distance
                best_taxi = i
            else:
                if (distance < best_distance):
                    best_distance = distance
                    best_taxi = i

    return best_taxi