Greedy Algorithm and Model for Analysis of a Bus Fleet's Operations

Duncan McGladdery

A project report in partial fulfilment of the requirements for the degree

Baccalareus (Industrial Engineering)

in the

Faculty of Engineering, Built Environment, and Information Technology

University of Pretoria

September 28, 2017

Executive Summary

Bus Company (BUSCO) approached Fourier-E, an industrial engineering company, to develop and optimise the Duty Master for their Sowetan operations. The Duty Master is a scheduling blue print that dictates the operations and management of a company's bus fleet. Through the manipulation of positioning routes and depot allocations, a Duty Master should minimise distances travelled and fleet size. In doing so, operating costs are reduced. Fourier-E had already developed their own algorithm for BUSCO's Sowetan Duty Master, which had resulted in substantial savings in the number of busses required to complete all revenue routes and total distances travelled. However, the run-time of their model of approximately 20 minutes to generate a solution is a serious limitation and prohibitive for client use.

The primary aim of this project was to develop an online user-friendly model to enable BUSCO to rapidly solve their fleet scheduling requirements. Specifically, the new algorithm had to be capable of generating a solution for the Duty Master in less than ten seconds. To accommodate the reduction in run-time, Fourier-E specified that the developed algorithm must generate a solution which is at least 80% as good as their existing one in terms of busses saved and positioning kilometres driven.

The problem of developing the Duty Master can be modelled as a Mulit-Depot Vehicle Routing Problem with Pickup and Delivery, Time Windows and Intermediate Facilities (MDVRPPDTWIF). To solve the problem, a greedy-heuristic was developed, called "Greedy-Bin". Computational tests showed that the algorithm exceeds all Fourier-E's specifications and performs particularly well in run speed, whch at 1.23 seconds, is 976 times faster than the existing approach. The Greedy Bin algorithm performs at 88% of Fourier-E's with regards to busses saved and 96.7% for kilometres saved. The Greedy-Bin algorithm met all validation criteria.

In order for clients to access the algorithm, an online user interface was developed which enables rapid evaluation of various operatonal scenarios. To achieve this, five variables that can be manipulated by the client were added to the model, namely: bus speed, revenue routes, loading buffer, distance buffer and day and night depot capacities. Model outputs are: the number of busses needed to complete all revenue routes, positioning distance, depot allocations and fleet utilisation throughout the day.

Manipulation of the Duty Master model also demonstrates its capacity to save on operating expenses by generating alternative solutions for depot allocations. Additionally, the financial implications of changing bus speeds and manipiulating loading and positioning buffers are demonstrated. Finally, in an industry where tenders for new revenue routes are highly competitive, the model can be used to assess the potential financial implications of adding new routes and in doing so, inform the decission of whether or not to tender for those routes.

Contents

List of Figures

List of Tables

List of Algorithms

Acronyms

AJAX	Asynchronous JavaScript And XML
BF	Best Fit
BFSP	Bus Fleet Scheduling Problem
BUSCO	Bus Company
BPP	Bin Packing Problem
CSS	Cascading Style Sheet
FF	First Fit
GA	Genetic Algorithm
GRASP	Greedy Randomised Adaptive Search Procedure
HVRP	Heterogeneous Fleet Vehicle Routing Problem
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
MDVRP	Mulit-Depot Vehicle Routing Problem
MDVRPPDTWIF	Mulit-Depot Vehicle Routing Problem with Pickup and Delivery, Time Windows and Intermediate Facilities
NF	Next Fit
NP	Non-Polynomial Time
OR	Operations Research
P	Polynomial Time
SATAWU	South African Transport and Allied Workers Union
TOWU	Transport and Omnibus Workers Union
VRP	Vehicle Routing Problem
VRPIF	Vehicle Routing Problem with Intermediate Facilities
VRPPD	Vehicle Routing Problem with Pickup and Delivery
VRPTW	Vehicle Routing Problem with Time Windows

Chapter 1

Introduction

This project produces an online model for a Mulit-Depot Vehicle Routing Problem with Pickup and Delivery, Time Windows and Intermediate Facilities (MDVRPPDTWIF) by developing an algorithm based on greedy logic to rapidly solve a large bus fleets scheduling requirements. In doing so it facilitates the optimisation of fleet size, positioning routes and depot utilisation.

1.1 Project Overview

Bus Company (BUSCO) is a South African bus company that operates countrywide. This project develops the Duty Master for its Sowetan fleet, which comprises of 984 routes and 430 buses that currently travel approximately 20,000km per day. Fourier-E is an industrial engineering consultancy company that was awarded the contract to optimise BUSCO's Duty Master. The Duty Master is the document that provides the blue print on how the bus fleet should operate and be managed. It dictates the revenue routes (transporting customers) that each bus must operate, the times that the busses need to be in position, the depots that the busses must start and end at, the rest depots that the busses must visit in the middle of the day and the routes that the busses must take to get into position, referred to as positioning routes. The high operating costs of the bus fleet, which includes fuel and maintenance cost and drivers' salaries, are influenced by the Duty Master. It is therefore important to develop a high quality Duty Master to minimise operating costs. Before the commencement of this project, Fourier-E had developed an algorithm capable of generating high quality Duty Masters for BUSCO. Their algorithm proved to be profitable by greatly reducing the kilometres driven and number of busses required to complete the daily fleet schedule; however, the run-time of approximately 20 minutes rendered the algorithm impractical for demonstration purposes to clients.

Following their successful completion of the initial contract, Fourier-E wanted to keep BUSCO engaged and to further develop their capacity as consultants to the bus transportation industry. Two potential opportunities for future contracts were identified. The first relates to optimising depot utilisation based on their location and capacities. The second is the need to have reliable data in order to analyse the implications of investing in new revenue routes. The aim of the project therefore was to develop a model that would enable Fourier-E to demonstrate to BUSCO and future clients, that using them as consultants to optimise their operating systems would be more profitable than the clients doing it in-house. To achieve this, a model was developed that allows the client to test the impact on their operating system of making changes to their depots, routes and loading and distance buffers. It was assumed that if the client could easily see the potential ben-

efits that would accrue from changing their operating system, then they would be more likely to contract Fourier-E to implement them. Requirements for the model were that it needed to be very easy to use and should require minimal configuration. In the long term, the model needs to be developed in such a way that a prospective client can do all the configurations themselves. However for this project, the aim was to create a model specifically for BUSCO that was preconfigured according to their specifications with regards to their route data (start and end locations, start and end times, distance of each route and route ID), depot capacities and their specific operational policies. For client-demonstration purposes, the model had to generate Duty Masters in near-real time while still producing high quality solution. The development of the model is the focus of this report.

1.2 The South African Bus Industry

As the South African bus industry is considered an essential service, it is subsidised and regulated by government (BUSCO, 2017). The government identifies routes and then makes them available for tender. Bus companies must then tender for the routes and prove that they have capacity to execute them. A limitation of government regulations of the routes is that there is very little flexibility in the start time of the routes. This makes it hard for the fleet manager to schedule the busses, as delaying a trip by half an hour or so is not an option.

Most of the bus drivers are affiliated with either South African Transport and Allied Workers Union (SATAWU), or Transport and Omnibus Workers Union (TOWU). The two unions have a lot of influence over the transportation industry. Their involvement also makes it quite difficult to retrench workers and change the workers' contracts. While acknowledging that it is important to look after the drivers, it adds complications when trying to implement a Duty Master that is likely to result either in some drivers no longer being required, or converting some contracts from full-time to part-time.

1.3 BUSCO and their Duty Master Specifications

BUSCO was founded over 60 years ago and is one of South Africa's largest bus companies, transporting more than 350,000 passengers per day. The company operates throughout South Africa. BUSCO operates a fleet of about 1,800 buses and has over 2,000 drivers, with busses travelling roughly 90 million kilometres a year! Due to the scale of their operations, small improvements to their Duty Master can have a large impact. BUSCO's management believes that their Duty Master is capable of improving as it was developed by manual methods. BUSCO has individuals that are skilled in doing this, but since their operations are so large and there are many constraints to consider, computer aided methods should be able to produce better results. This project demonstrates that this is indeed the case.

Optimisation of the Duty Master was evaluated against two criteria, making the project multi-objective. Firstly, the developed model needed to reduce the number of busses required to complete the fleet schedule. Secondly, the model needed to be able to reduce the total distance driven by the fleet. The client had specified that the reduction of busses had to be prioritised over the reduction of kilometres driven. Reducing the number of busses used will substantially reduce the operational and capital costs. Every bus requires a driver, regular maintenance and costs about one million rand to purchase (BUSCO, 2017). Additionally, the model needed to efficiently allocate busses to day and overnight

depots, which would result in a further reduction in the total distance driven. Reducing the amount of kilometres driven will save in bus maintenance and fuel. A further benefit of an efficient bus schedule is the reduction in pollutants. Fossil fuels are a major contributor to greenhouse gas emissions, which can be reduced by minimising distances travelled.

Unfortunately reducing busses will result in some drivers losing their jobs. This is not a desired outcome but it can be argued that the leaner a company is, the lower the cost they can sell their product or service to society. It is hard to predict if BUSCO will lower their prices or if the leaner operations will increase the job security of the other drivers. For these reasons and others that will be discussed later, this project will not discuss utilitarian or capitalist philosophies, but will acknowledge the fact that the results of a successful project could negatively affect some individuals, specifically bus drivers and bus maintenance teams.

1.4 Problem Statement

The research question that this project answers is how can we develop a user-friendly online model to rapidly solve a large bus fleet's scheduling requirements in terms of optimisation of fleet size and distance travelled. The bus fleet's scheduling requirements will be referred to as the Bus Fleet Scheduling Problem (BFSP) and its solution is an optimised Duty Master. The problem arose in response to the consultancy company, Fourier-E, wanting to retain BUSCO and attract future clients in the bus transportation industry, by providing them with an easy to use online interface that would enable manipulation of a set of parameters in order to optimise their Duty Masters. Although Fourier-E had already developed an algorithm to facilitate this, the run-time of approximately 20 minutes rendered it impractical in terms of being integrated into an online-tool.

The development of a fast algorithm capable of generating a solution in less than ten seconds, to address the slow run-time problem of Fourier-E's algorithm, and the development of the online user interface are the primary objective of this project. The integration of these two objectives results in the Duty Master Model.

To solve the BFSP and hence optimise fleet scheduling in terms of minimising the number of vehicles and distance travelled, two costs needed to be minimised, namely: the fixed costs of bus purchase and associated driver salaries, collectively referred to as bus ownership costs, and the variable costs associated with getting each bus into position in order to execute the revenue routes, which consists of fuel and maintenance costs. The model parameters which could be manipulated by the client were identified as: bus speed, revenue routes, allocation to the day and night depots, and a loading and a distance buffer. The buffers allow for unforseen delays that could result from complications with people getting onto busses, or traffic jams and road works that could impact on the time to complete the route. The model constraints are existing depot capacities and the feasibility of route sequencing.

1.5 Research Design and Methodology

The main deliverable of this project was the development of a user friendly online tool that enables the optimisation of a bus fleet's Duty Master. This deliverable is called the Duty Master Model.The model developed had to allow clients (in this case, BUSCO) to manipulate their operating procedures, and in doing so generate new Duty Masters which then could be evaluated against their existing operations. Operating procedures that the client can change though the interface that was developed are: bus speed, loading buffer,

distance buffer, day and nighttime depot capacities and revenue routes. The model's outputs are: number of busses needed to complete all revenue routes, positioning distances, depot allocations and fleet utilisation throughout the day.

In order to create the model for this project, an algorithm was developed and then a user interface was built on top of it. While it may have been preferable to use the algorithm that Fourier-E had already developed, a few problems prevented this. Firstly, Fourier-E's algorithm takes on average 20 minutes to generate the solution. This is an unacceptably long time for the model as clients expect to see results without delay. The second issue is that Fourier-E will not allow their algorithm to be saved on any devices other than their own. They are also not willing to allow remote access to their servers. Bearing these constraints in mind, Fourier-E specified that they wanted a new algorithm developed with a run-time of 10 seconds or less. It was expected that the new algorithm would not deliver results as good as Fourier-E's algorithm owing to it having to run so much faster and thus some solution quality compromise was made. Fourier-E set a lower specification limit, namely that the new algorithm must result in savings of at least 80% of that achieved by their existing algorithm. The aim of this project was to develop the faster algorithm and imbed the algorithm in an online-tool, to be used by potential clients.

In order to design the model, the methodology presented by Manson (2006) was followed. Manson divides the methodology into five phases:

1. **Awareness of the Problem**. Before the problem can be solved it is vital to fully understand the problem. For this project, the problem that needed to be solved was how can a rapid and user friendly online model be developed that would enable the manipulation of a set of parameters to optimise bus fleet scheduling in terms of minimising costs by optimising positioning routes and depot utilisation.

2. **Suggestion**. This phase involves suggesting a potential solution. In order to do this, a study of previous methods used to solve related problems was undertaken. In particular, reference to Vehicle Routing Problems and Bin Packing Problems were studied to provide a foundation for the algorithm. Fourier-E had stipulated that the model needed to be available online and therefore the need to consider alternative interfaces, such as a desktop application, was not necessary.

3. **Development**. This is the phase when the proposed solution is developed. Specifically, the algorithm and the online interface that takes the user inputs, generates the Duty Master, and displays the potential benefits of making changes to the client's operations, were developed.

4. **Evaluation**. The solution needs to be scrutinised to make sure it is performing as it should. In this project, the developed algorithm was evaluated against the performance of Fourier-E's existing algorithm in terms of run-speed, number of busses used and positioning kilometers of the fleet. Additionally, results were evaluated through visual validation of the Duty Master in the form of a graph; the scrutiny of the operations of 19 busses to test for violation of any constraints, and through a sensitivity analysis of model performance whilst varying input parameters.

5. **Conclusion**. During this phase of the project, opportunities in BUSCO's current fleet's operations were identified in terms of optimising the allocation of buses to depots; identifying times during the day when new revenue routes could be added without requiring additional busses; and finally, suggesting future projects to identify optimum locations for depot establishment.

As part of the model validation, specific what-if scenarios were analysed. The effect of changing depot allocations and capacities on total kilometres driven was investigated. The impact of increasing bus speed and reducing buffer times on reducing the number of busses needed and total kilometres driven was demonstrated. Finally, the impact of adding new revenue routes on total busses required and kilometres driven, as well as the model's ability to identify the most feasible time of the day to add new routes, were analysed.

1.6 Document Structure

In the next chapter the literature is reviewed in order to establish the type of model that needs to be developed and the appropriate approach to solving the BFSP. Different optimisation methods are presented and the benefit of conceptualising the problem as a Vehicle Routing Problem (VRP) and Bin Packing Problem (BPP) is discussed. The focus of Chapter 3 is the development of a rapid algorithm and user interface which together produce the Duty Master Model. Chapter 4 demonstrates the practical application of the developed model as a tool to aid decision making under various scenarios through the manipulation of the model's parameters. In the final chapter of this project, the developed model is assessed and potential future projects are presented.

Chapter 2

Literature Review

The literature review commences with a discussion of the importance of modelling and the classification of models according to their intended use, in order to identify the type of model that needs to be developed for this project. The conditions for solving the Vehicle Routing Problem (VRP) as a Bin Packing Problem (BPP) are then presented. How the Bus Fleet Scheduling Problem (BFSP) differs from classical VRP is discussed, leading to the problem being classified as an Mulit-Depot Vehicle Routing Problem with Pickup and Delivery, Time Windows and Intermediate Facilities (MDVRPPDTWIF). The Greedy Randomised Adaptive Search Procedure (GRASP) metaheuristic is reviewed as a method to sole the research problem. The chapter concludes with an analysis of Fourier-E's existing algorithm which sets the benchmark against which this project's algorithm was developed.

2.1 The Importance of Modelling and Model Use

The article by Pidd (2010), *"Why modelling and model use matter"*, provided the foundation for investigating the requirements of the project's model. According to the article, an Operations Research model is an "external and explicit representation of part of reality as seen by the people who wish to use that model to understand, to change, to manage and to control that part of reality" (Pidd, 2010, p. 14). The initial role of the modeller is to identify the part of reality that will be modelled. In this project, the reality that needed to be modelled was BUSCO's Sowetan fleet operations under different route and depot parameters. To achieve this, the model needed to predict how many busses and kilometres would be saved when the above mentioned parameters were changed.

The traditional method of illustrating models is with white, grey or black boxes that represent the process being modelled (Pidd, 2010). A white box represents a process that is fully understood by the modeller. In contrast, a black box represents a process that the modeller does not understand at all. A white box process can be validated by scrutinising the model's internals (for example: equations or logic). Black box processes are validated by examining the outputs whilst controlling the inputs.

According to Pidd, a limitation of earlier modelling methodologies was that they tended to be developed for specific cases. Overcoming this limitation, he proposed a framework that is independent of the nature of the model and that can thus be applied in many different circumstances. The underlying principle of Pidd's framework is that it is based on intended model use; that is, the decisions and processes that the models are intended to support and, in particular,"the ways in which people will interact with those models" (Pidd, 2010, p. 14). According to Pidd's framework, four model archetypes exist, based on the purpose for which they are used. An understanding of these archetypes, combined with

BUSCO's needs, was necessary in order to understand the project model's requirements.

2.1.1 BUSCO's intended model use

The model developed for this project will be used by BUSCO to test the effect of adding new routes, changing the bus allocations to their depots, changing depot capacities, changing bus speeds, and reducing the distance buffer and loading buffer. BUSCO will very rarely want to change the capacities of their depots and will probably only add or remove routes a few times a year because, as mentioned earlier, the government decides if new revenue routes should become available. This means that BUSCO will not use the model on a regular basis but when they do, it needs to be fast and easy to use so that multiple experiments can be run. By BUSCO using the model to identify the effects of various manipulations of their operating procedures, Fourier-E is confident that the model will encourage BUSCO, and future clients, to contract them to develop new Duty Masters incorporating those changes.

2.1.2 Pidd's framework of model use

To demonstrate how models are used, Pidd (2010) develops a framework based on two opposing axes. The axes represent the frequency that the model will be used and the level of effort required to develop the model. Different types of model-use exist along the spectrum of the two axes. Figure 2.1 is a visual representation of the spectrum. Pidd divides the spectrum into four model archetypes: **Decision Automation, Routine Decision Support, Investigation & Improvement** and **Provide Insight**. He provides information that makes it easy to classify models into one of these four types according to intended use and amount of effort required to develop the model. Each type of model serves a different purpose. Models for **Decision Automation** generally require a lot of time to set up as it is important that they are highly accurate. On the other end of the spectrum are models for **Providing Insights**.

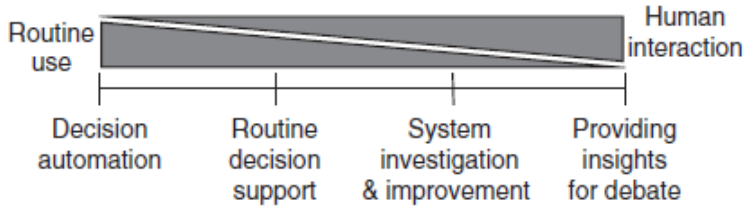

Figure 2.1: A Spectrum of Model Use. (Pidd, 2010, p. 16)

Referring to the intended use of the BUSCO fleet scheduling model, two of Pidd's archetypes were eliminated, namely: modelling for **Decision Automation** and modelling for **Routine Decision Support**. Modelling for **Decision Automation** was eliminated as the decision to contract Fourier-E is decided by the client not the model. Since the model will not be used on a regular basis, modelling for **Routine Decision Support** was eliminated as the model's use is not routine. The remaining two model-use archetypes, namely: **System Investigation & Improvement** and **Providing Insights**, needed to be investigated before a final decision could be made regarding which to use for this project as it was not clear which one of those types the BUSCO model fell under.

According to Pidd (2010), modelling for **System Investigation & Improvement** is the most common model archetype for Operations Research. He explains that models of this type are used to support investigations that are relatively unique, which may involve "system design, system improvement or just an attempt to gain understanding of a very complex situation" (Pidd, 2010, p.18). He argues that these types of models are often used for a short period of time as they are generally developed to investigate a very specific problem. They are used in Operations Research extensively because they provide a structure that allows investigators to better understand the system being modelled and allows them to perform what-if analyses. The ability to facilitate what-if analyses makes them very useful for system improvement as the scenario can be investigated before large amounts of capital are invested. Big decisions are made based on the findings of these models, so it is important that they are accurate. Pidd highlights that the challenge of validating **System Investigation & Improvement** models is that it is often impossible to compare the outputs of the model to actual data because the scenario being tested does not yet exist. The way that this problem is minimised, is by doing white box validation in which the internals of the model are extensively scrutinised.

Modelling to **Provide Insights** is used for situations that have been termed as wicked problems. Wicked problems are those where there are two or more stakeholders and all have different but legitimate ideas on how to solve the problem (Pidd, 2010). The purpose of developing a model to **Provide Insights** is to create a structure that allows all parties to see the perspective of the other parties with the end goal of facilitating consensus between all parties on what course of action should be taken. Models that facilitate this are often classified under soft Operations Research as they are based more on human aspects than data. That being said, there are quantitative models that are classified in this archetype. This model type is the least intensive to develop as it is expected that the model users will make the bulk of the decisions and that the model is just there to facilitate the process.

A further consideration of model development is the type of input data required. Collecting data is a time consuming process as it often involves communication between people or machines and, since most models are data sensitive, formatting the data needs to take place. Pidd (2010) explains that the more weight the model has on the final decision, the greater the importance of large amounts of accurate data. As a result, models for **Decision Automation** are the most data intensive, whilst models which **Provide Insights** are the least. For modelling to **Provide Insights**, Pidd allows for a relaxation of data accuracy as the users have a lot of say in making the decision and there are often cases where accurate data is simply not available.

2.1.3 Classifying the project's model

Having outlined the differences between the **System Investigation & Improvement** and **Providing Insights** archetypes, the BUSCO model could be classified. A criterion for the model was that it be configured to allow the client to do what-if analysis on scenarios such as adding routes or changing the location or capacity of depots. The ability to do these investigations would suggest that the model should be classified as a model for **System Investigation & Improvement**. However this was not the case, as the main use of the model is to entice the client to contract the consultant, Fourier-E, to develop the new Duty Master or to investigate further the effect of changing parameters. Bearing this functionality in mind, Fourier-E stressed that the intention of the model was just to demonstrate an estimate of what Fourier-E think they could achieve if they did a full Operations Research investigation. Therefore, the model that was developed will **Provide Insights** to encourage BUSCO to contract Fourier-E to undertake a deeper investigation,

rather than the client attempting to solve the problem by themselves.

2.1.4 Model validation

Validating models is crucial because if users do not trust the output they will not use the model. There are many methods that can be applied to validate models and these methods are often customised to the specific problem; however, they normally apply some type of white or black box validation. When deciding how the model would be validated Pidd explains: "There seems to be general agreement that, in practice, fitness for purpose is the main focus of validation efforts and also a realisation that no model will ever be wholly valid, except within defined circumstances" (Pidd, 2010, p.19).

When models are used to answer what-if scenarios that do not currently exist the models tend to be harder to validate. The general assumption is that if the model works for the as-is configuration of the process, it should provide a reasonably good solution for what-if scenarios. Apart from the simplest of cases, this is a weak assumption and Pidd (2010) suggests that the complexity of models should be investigated and the more complex they are, the greater is the need to scrutinise the outputs of the model under controlled inputs. This results in the process being represented by a dark grey box. Understanding what makes a model complex is not as straight forward as one would think. At face value one could assume that complexity is just a matter of scale, but this neglects the interactions that components have with each other and the cognitive abilities of the modeller. To clarify the issue, Pidd (2010) mentions three definitions of complexity and then brings them together by defining a complex model as one that this is likely to be complicated, hard to understand and model, and may require multiple representations. It is important to understand the complexities of the system because there is a trade-off between accuracy of the solution and the time it takes to run the model. Due to the inherent complexity of most processes, it is rarely viable to develop a model that is 100% accurate. For this project, Fourier-E set a lower bound for accuracy for the new model at 80% of their existing algorithm. What this means is that they were prepared to accept a 20% drop in quality of solution in favour of a model that is fast and easy to use. Fourier-E's specified solution time of 10 seconds, compared with their current soluntion time of 20 minutes, clearly demonstrates the unfeasiblity of their existing algorithm for client demonstration purposes. The practice of establishing trade-offs in model development was commented on by Box et al. (1987, p.424) who boldly stated: "essentially, all models are wrong, but some are useful".

The project model's fitness was evaluated according to how quickly and accurately it could solve BUSCO's fleet scheduling problem compared with Fourier-E's existing algorithm's solution. Pidd mentions that in most circumstances validating models to **Provide Insights** is problematic due to the bulk of them being based on soft data. This was not the case for this project as Fourier-E had provided the output of their algorithm. This allowed for a black box analysis whereby the output of the model in terms of minimising the number of busses needed and kilometres driven to execute BUSCO's fleet schedule could be compared to that of Fourier-E's existing algorithm.

Ultimately all models have to add value, in fact they need to add more value than the effort required to make them. The main purpose of modelling for insight is to help two parties understand each other. This aligned very well with Fourier-E's objective. If clients have a good idea of what Fourier-E can achieve and the model's solution seems better than what the client thinks they can achieve in-house, then the client is likely to contract Fourier-E to optimise their Duty Master.

Having discussed the performance requirements of the model, the next challenge of the project was to create an algorithm that could achieve those requirements.

2.2 Vehicle Routing Problem

In order to develop the algorithm it was important to first understand the type of problem that the algorithm needed to solve and the methods that have been used to solve those types of problems in the past.

The project's BFSP is a variant of the well studied field of VRP. The classical VRP aims to minimise the distance travelled by a fleet of vehicles as they make deliveries to customers. There are many variants of the VRP and it is important to identify which variants are present in the BFSP. Applying the classification used by Braekers et al. (2016), the four variants of the VRP that are present in the BFSP were identified as follows:

1. In the classical VRP the customers are represented by nodes that have a single location. In the BFSP this is not the case, because when passengers travel on a bus they are moved from one location to another. This variant is known as the Vehicle Routing Problem with Pickup and Delivery (VRPPD).

2. The classical VRP has a single depot. This project's problem however, deals with multiple depots. When there are multiple depots the problem is called a Mulit-Depot Vehicle Routing Problem (MDVRP).

3. A further attribute of the BFSP is that each route has a fixed start time, so each bus needs to be in position to load passengers before the start time. This introduces a time constraint and these problems are referred to as the Vehicle Routing Problem with Time Windows (VRPTW).

4. Finally, each bus needs to visit a depot in the middle of the day so that the driver can rest. This adds another variant called the Vehicle Routing Problem with Intermediate Facilities (VRPIF).

Bearing these variants in mind, BUSCO's BFSP can be classified as a MDVRPPDTWIF.

A further distinction of the project's problem compared with classical VRP, is that unlike classical VRP where there is only a single objective to minimise distance, in the BUSCO Duty Master problem there are multi-objectives with the primary goal being to to minimise bus numbers. To solve this, the Bin Packing Problem was considered.

2.3 Bin Packing Problem

Martello and Toth (1990, Chap 8) describe the BPP and methods that have been developed to solve it. The objective of the BPP is to fit objects into as few bins as possible. By conceptualising a bin as being a vehicle and the objects with varying volume as the customers with varying time window constraints, the VRPTW can be modelled as a BPP. There are three common approximation strategies used for this. They are Next Fit (NF), First Fit (FF) and Best Fit (BF). Next Fit adds the objects into the current bin until the bin does not have capacity to add another object. At that point a new bin is opened and filled until yet again no more objects can fit and then a new bin must be opened. The process continues until all objects have been placed in bins. The problem with this method is that as soon as a new bin is opened the previous bins are never looked at again and since the objects can be of varying sizes, it is often possible to place an object in a bin

that is not evaluated. The First Fit method was developed to solve this exact problem, because it first tries to place the object in the first bin, but if it cannot fit there, it then moves onto the next and so on until it finds a bin into which it can fit. Only after looking at all the bins will a new bin be opened if the object could not be placed. The Best Fit method uses very similar logic to the FF with the difference being that the BF places the object in the bin that will have the smallest available capacity after the allocation, whereas the FF places the object in the first available bin. The logic of BF is presented in Algorithm 1

Algorithm 1: Best Fit Algorithm For BPP

Input : Objects with varying volume ***Obj***
Output: Solution $\boldsymbol{S}^*$

for *object in* ***Obj*** **do**
  if *object can be placed in an open bin* **then**
    find bin that will have the smallest available capacity after allocation;
    allocate object to the found bin;
  else
    open new bin;
    allocate object to new bin;
$\boldsymbol{S}^* \leftarrow$ compile solution;

return $\boldsymbol{S}^*$

The performance of the NF, FF and BF can be improved by sorting the objects in descending order based on their size. Sorting the objects in this manner results in the methods being renamed to NFD, FFD and BFD where the D represents the word decreasing. All of these methods are constructive and deterministic so can rapidly produce solutions that can be validated through white box analyses. Both of these traits were considered ideal for achieving the project's model requirements of being fast and user friendly.

2.4 Solution Methods

A complete solution method for the variant of this project's VRP has not been found. Braekers et al. (2016) revealed that in recent years 71.25% of VRP have be solved with metaheuristic, 17.13% with exact methods and 9.79% with classical heuristics. All the aforementioned methods to solve the BPP are classified as classical heuristics. Figure 2.2 is Braekers et al. (2016)'s summary of solution methods. The three approaches to solution methods were studied in order to select an appropriate method to solve BUSCO's fleet scheduling problem.

Applied method (1.2)	Number of models	Relative presence							
		Overall (%)	2009 (%)	2010 (%)	2011 (%)	2012 (%)	2013 (%)	2014 (%)	2015 (%)
Metaheuristic (1.2.3)	233	71.25	65%	63	65	77	80	76	65
Exact Method (1.2.1)	56	17.13	17	20	26	10	13	17	19
Classical Heuristic (1.2.2)	32	9.79	11	15	15	17	4	5	10
Real-time solution methods (1.2.5)	11	3.36	6	0	6	4	5	2	0
Simulation (1.2.4)	7	2.14	4	4	6	0	2	0	0

Figure 2.2: Overview of applied methods in absolute and relative numbers. (Braekers et al., 2016)

2.4.1 Exact Methods vs Metaheuristics

Talbi (2009)'s book titled *"Metaheurisitics from Design to Implementation"* stresses that it is important to understand the type of problem that is to be solved before any effort is put into solving the problem. The reason for this is that many people make the mistake of jumping to metaheuristics to solve problems that can be solved with exact methods. Exact methods provide a solution in optimality. What this means is that the global optimal solution has been found. The alternative to exact methods is approximate methods. Metaheuristics fall under this catagory. For these methods there is no way to guarantee that the algorithm will provide a solution in optimality, unless the problem can also be solved with exact methods or there is no degree to which various solutions can be measured. An example is an algorithm that solves a large Sudoku: the answer to a Sudoku is either right or wrong, so no solution can be considered better than another. Talbi (2009) provides an overview of the various methods to solve optimisation problems which are illustrated in Figure 2.3

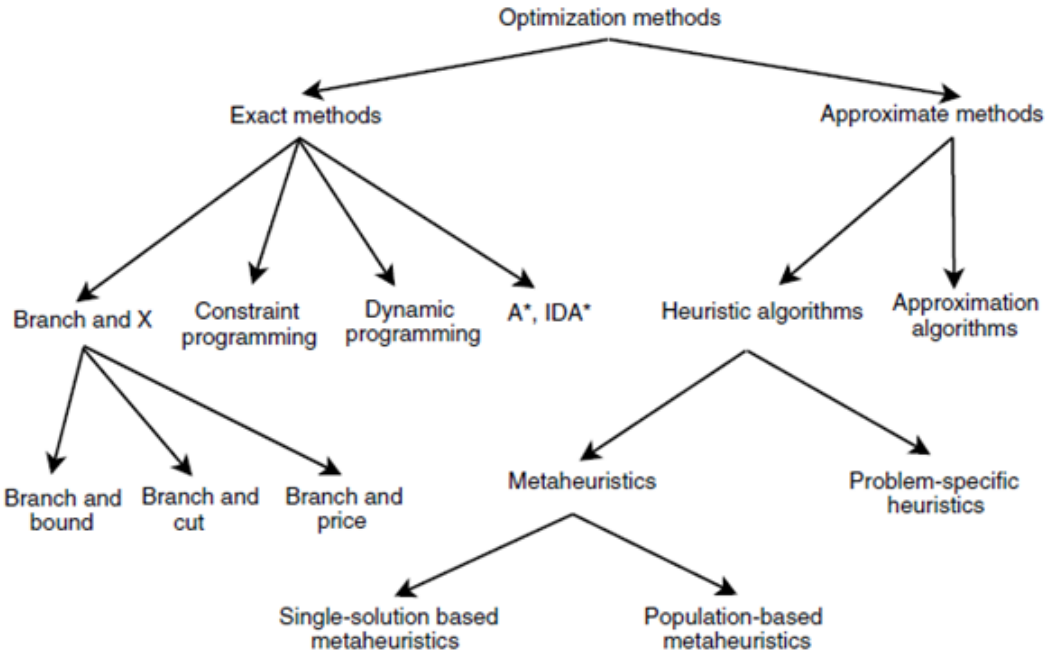

Figure 2.3: Optimisation Methods (Talbi, 2009, p. 18)

Apart from a few cases where exact methods take too long and the quality of solution is not a top priority, all problems that can be solved with exact methods should be. Approximate methods should only be used when exact methods will not work.

In order to make the initial selection to use exact methods or approximate for this project, it was first necessary to classify the problem being solved as Polynomial Time (P) or Non-Polynomial Time (NP). The main distinguishing factor between these two types of classifications is how well the algorithms developed to solve them function when the size of the problem increases. P class problems scale very well as the complexity of the problem follows some polynomial as the size increases, whereas NP problems grow exponentially. The result of this exponential growth makes NP problems unsuitable for exact methods once the size of the problem is no longer small. There is extensive documentation available that helps classify problems as P or NP (Talbi, 2009). If the classification of the problem

being solved is not documented, an investigator can apply complexity theory to make the classification. A further distinction can be made within the NP class. If it is impossible to prove in polynomial time that the solution to the problem is the global optimal, it is considered to be an NP-hard problem. Most VRP problems are considered to be NP-hard. Sudoku can also be used to illustrate the difference between NP-hard and NP. Solving a Sudoku is considered to be NP because if the solution to a Sudoku is given, it is quite simple to prove that there is no better solution. In comparison, it is not possible to prove this for most VRPs, hence they are considered NP-hard. There are some cases where small VRP problems can be solved with exact methods but these are rare and were not applicable to this project. An understanding of P and NP is vital to participate in discussions about optimisation as erroneously classifying the type of problem will result in the application of inappropriate solution methods. Due to the scale and number of variants in BUSCO's BFSP, there was no way to say with certainty that the optimal solution to the problem could been found and therefore it was classified as NP-hard. As a result exact methods were not applicable so are not discussed further.

On a side, not proving or disproving that P=NP is one of the seven millennial problems and carries a prize of one million dollars (Claymathn, 2017). In essence, what P=NP implies is that if it is easy to validate a solution to the problem then it is easy to create the solution. Most people believe that P is not equal to NP but if it can be proven it will simplify many of the worlds optimisation problems and take away the skill that is generally required to provide answers to complicated problems. Aaronson (2006) explained the implications of P=NP as: "There would be no fundamental gap between solving a problem and recognising the solution once it is found, everyone who could appreciate a symphony would be Mozart, everyone who could follow a step by step argument would be Gauss".

2.4.2 Metaheuristics vs Classical Heuristics

Metaheuristics are a higher level methodology that allows for a broad range of problems to be solved whilst heuristics are often developed to solve a specific problem. Both are appropriate when the nature of the problem to be solved requires approximation methods. According to Braekers et al. (2016), metaheuristics can cater for none optimal local selections in the development of the solution whereas classical heuristics always make local optimal selections. This results in metaheuristics being able to explore more possible solutions which normally lead to a better solution than that of a classical heuristic. This better solution is at the expense of the algorithm's run time. The difference in run time between metaheuristics and classical heuristics is attributed to the fact that for the vast majority of cases, metaheuristics are iterative meaning they create and manipulate multiple solutions until a stop criterion is met, whereas classical heuristics are constructive resulting in the construction of a single solution. Without some form of greedy logic whereby local optimal selections are made at each step in the construction, classical heuristics tend to be limited in their use.

Another attribute of metaheuristics that needed to be considered was whether the resulting algorithm used deterministic or stochastic methods. A deterministic algorithm will always output the same solution if its inputs are the same, whereas a stochastic algorithm will incorporate randomness into the process so the output will not always be the same. The benefits of adopting a stochastic approach for the Duty Master Model are demonstrated by reference to Kontoravdis and Bard (1995)'s GRASP metaheuristic.

2.4.3 GRASP Metaheuristic

The paper by Kontoravdis and Bard (1995), "A GRASP for the Vehicle Routing Problem with Time Windows", develops a GRASP and proves that it is effective at providing good solutions to the VRPTW and multiple objectives. In their paper, a brief description of the VRPTW is presented and then described with graph theory.

Kontoravdis and Bard (1995) present a VRPTW that is different to the classical one, as it has two objectives instead of just one. The primary objective is to minimise the required number of vehicles to make all the deliveries or pickups and the secondary objective is to minimise the total distance travelled. Their version of the problem is more applicable to many real world problems than classical VRPTW and specifically mentions bus scheduling, rendering it particularly insightful for BUSCO's BFSP. The authors developed a GRASP metaheuristic because it produced good solutions quicker than other metaheuristics.

The GRASP metaheuristic combines a greedy heuristic with randomisation. In the development of the solution, the cost of adding a revenue route to each of the available vehicles is calculated and stored as cost data. A standard greedy heuristic will look at the cost data and allocate the route to the vehicle with the lowest associated cost. This results in a deterministic output. In comparison, a random heuristic will look at the data and randomly select an available vehicle from the list and in doing so will rarely select the vehicle with the lowest associated cost. This results in the output being stochastic instead of deterministic. The stochastic nature allows for the construction of the solution to be iterated over a set period, or until a criterion is met. Once the iteration has stopped, the best solution of the process is selected. This adds the metaheuristic component. The method allows for a greater number of solutions to be explored but is very inefficient for large problems. The GRASP metaheuristic is the amalgamation of these two methods. The GRASP metaheuristic looks at the cost data and then randomly selects from only the best options. Kontoravdis and Bard (1995) found that selecting from the top three options produced the best results. The logic of the GRASP metaheuristics is shown in Algorithm 2

Algorithm 2: GRASP for VRP

Input : Customers $\boldsymbol{C}$,Iteration Limit t_{max}.
Output: Solution $\boldsymbol{S}^*$

$t \leftarrow 0$;
$\boldsymbol{BestCost} \leftarrow VeryBigNumber$;

while $t \leq t_{max}$ **do**
  for *customer in* $\boldsymbol{C}$ **do**
    find three closest vehicles to customer;
    randomly allocate customer to one of these three vehicles;
  $\boldsymbol{S} \leftarrow$ compile solution;
  $\boldsymbol{Cost} \leftarrow$ evaluate cost of solution;
  if $\boldsymbol{Cost} \leq \boldsymbol{BestCost}$ **then**
    $\boldsymbol{BestCost} \leftarrow \boldsymbol{Cost}$;
    $\boldsymbol{S}^* \leftarrow \boldsymbol{S}$;
  $t \leftarrow t + 1$;
return $\boldsymbol{S}^*$

The approach by Kontoravdis and Bard (1995) demonstrates that it is very important

to order the customers before they are inputted into the algorithm as the order can have a large effect on the performance of algorithms that incorporate greedy logic. Algorithms based off greedy logic are prone to deteriorating solution quality as options to choose from get exhausted. As a result it is important to input the customers in a logical manner to try and mitigate this effect. Kontoravdis and Bard (1995) mention that ordering the customers by their distance from the depot, or in ascending order of their time window constraints, produces good results. Applying the GRASP to the BFSP problem, the customers represent the revenue routes that need to be optimally allocated to the fleet's busses.

The fact that the at the core of a GRASP is a greedy heuristic that identifies the top few allocation options presents a unique opportunity. If the algorithm developer stops before including the iterative functionality and function that randomly selects from the top allocation options and instead adds functionality to only select the best option, the result will be a greedy algorithm instead of a GRASP. Greedy algorithms are classical heuristics and a GRASP is a metaheuristic. The unique opportunity is that the frame work of a GRASP makes it relatively easy to convert a classical heuristic into a metaheuristic and vice versa.

2.5 Fourier-E's Genetic Algorithm

Fourier-E initially tried to manipulate a JSPRIT package to solve the routing problem. JSPRIT is an open source package that has been designed by top computer scientists to solve VRP problems (JSPRIT, 2017). The software package allows for the code to be manipulated. This method was aborted by Fourer-E as it was too time consuming. JSPRIT is written in Java which is a common language but relatively hard to follow by Fourier-E's engineers. Fourier-E's engineers have been using Python over Java and since VRP problems are so well documented, they decided to build the algorithm from scratch. In addition to this, developing a Python based solution mitigates the risk of employees that are involved in the project leaving as there are six engineers at Fourier that are skilled with Python. They chose the Genetic Algorithm (GA) as the metaheuristic to develop the Duty Master. The GA is a nature inspired, iterative, population-based and stochastic metaheuristics (Talbi, 2009). As discussed previously, although their model achieves the objective of reducing bus numbers and kilometres driven to execute the BUSCO fleet schedule, a serious limitation of their Genetic Algorithm (GA) is that it takes about 20 minutes to generate a solution..

2.6 Conclusion

The importance of modeling and the framework for classifying models according to their intended use that was proposed by Pidd (2010) has been presented. The model to be developed for this project has been identified as a model to **Provide Insights** as it is intended to enable the client to manipulate certain parameters of their operating procedures in order to demonstrate potential improvements which could be made to their their Duty Master.

The role of exact methods, metaheuristics and classical heuristics in problem solving has been discussed. Referring to the work of Talbi (2009), the research problem was classified as a fair sized NP-hard problem, which ruled out exact methods and suggested that metaheuristics or heuristics were the most appropriate method for addressing the research problem.

It was identified that with a few adaptations BPP logic can be used to achieve the primary objective of reducing the number of busses needed to complete all revenue routes.

The various stages of the development of a GRASP metaheuristic where examined and it was identified that the partial development results in a greedy algorithm which is a classical heuristic, so has the advantage of a fast run time. Whereas the complete development of GRASP metaheuristic has the advantage of a better solution. Fourier-E has used the Genetic Algorithm to solve the BUSCO fleet scheduling problem, which works well in terms of reducing costs but takes a long time to run.

Drawing on the information presented in this literature review, three options for algorithm development for the research problem became apparent. The first was to partially develop a GRASP to the point where it is a classical heuristic. The second was to fully develop the GRASP metaheuristic. The third was to follow Fourier-E and use the Genetic Algorithm. All of these options will include aspects of bin packing logic in order to reduce the busses. These options are considered in the next chapter.

Chapter 3

Development of the Duty Master Model

The objective of the model to be developed was to provide BUSCO with an online tool that would demonstrate the effect of making changes to various aspects of their operating procedures. Additionally, it was intended that the model would also illustrate that Fourier-E can generate much quicker solutions and a greater variety of solutions than the client could generate in-house. The more frequently clients make changes to their operating procedures, the more opportunities there are for Fourier-E to sell an updated Duty Master. In order to achieve this, an algorithm that generates a Duty Master in less than ten seconds and provides a solution that is at least 80% as good as Fourier-E's existing GA needed to be developed and incorporated into a user interface.

3.1 Defining the Problem Mathematically

The model's parameters that are available for client manipulation were defined as: bus speed, revenue routes, allocation to the day and night depots, and a loading and a distance buffer. The model constraints are the client's existing depot capacities and the feasibility of their route sequencing. The model outputs are the optimised number of buses, positioning distances, depot allocations and fleet utilisations throughout the day, which demonstrate the effective savings that could result from implementation of the Duty Master generated by the model.

The Duty Master model developed for this research project is an amalgamation of the algorithm and the user interface.

In order to define the research problem mathematically, the following needs to be stated:

- Let the cost of owning a bus be represented by $\boldsymbol{C}_1$.
- Let the cost of fuel and maintenance per kilometre be represented by $\boldsymbol{C}_2$.
- Let x be the number of available busses in the fleet.
- Let each available bus be represented by k where $k \in \{1, 2, 3, \ldots, x\}$.
- Let the total distance that each bus travels to get into position and to execute its revenue routes be represented by P_k.

Then with the decision variable x, the objective function can be modelled as:

$$\min z = x\boldsymbol{C}_1 + \boldsymbol{C}_2 \sum_{k=1}^{x} \boldsymbol{P}_k. \tag{3.1}$$

In order to calculate P_k the operations that each bus undertakes needs to be explained. Every day of the week each bus does the following:

1. Travel from its overnight depot to its first revenue route.
2. Execute its first revenue route.
3. Get into position then execute its next revenue route.
4. Repeat step three until all its morning revenue routes have been executed.
5. Travel from the end location of the last morning revenue route to its day depot.
6. Travel from its day depot to its first afternoon revenue route.
7. Execute the first afternoon revenue route.
8. Get into position then execute its next revenue route.
9. Repeat step eight until all its afternoon revenue routes have been executed.
10. Travel from the end location of its last revenue route to the same overnight depot that it started at.

The smaller the value of P_k the better is the value of the objective function so the goal is to execute all the above steps in as few kilometres as possible without violating any of the constraints. In order to mathematically express the calculation of P_k we need to state the following:

- Let $R_{k,i}$ be the notation for the i^{th} revenue route that is assigned to bus k. For example $R_{2,4}$ is the route that is assigned to the second bus and it is the 4th revenue route that the bus will execute in its tour.
- Let $\boldsymbol{D}(i, j)$ be the distance between location i and location j. i and j are the start and end locations of all revenue routes, and the available depots.
- Let LM_k be the index of the last morning revenue route that was assigned to bus k.
- Let LA_k be the index of the last afternoon revenue route that was assigned to bus k.
- Let the set of depots be represented by d such that $d \in \{1, 2, 3, 4, 5, 6, 7\}$. There are seven depots available for busses to be allocated to.
- Let $DF_{d,k}$ be the day depot d (intermediate facility) that was assigned to bus k.
- Let $NF_{d,k}$ be the night depot d that was assigned to bus k.

With the above defined the calculation of $\boldsymbol{P}_k$ is as shown in the equation below:

$$\begin{aligned} P_k = \boldsymbol{D}(NF_{d,k}, R_{k,1}) + \left(\sum_{i=1}^{LM_k - 1} \boldsymbol{D}(R_{k,i}, R_{k,i+1}) \right) + \boldsymbol{D}(R_{k,LM_k}, DF_{d,k}) \\ + \boldsymbol{D}(DF_{d,k}, R_{k,LM+1}) + \left(\sum_{i=LM_k+1}^{LA_k - 1} \boldsymbol{D}(R_{k,i}, R_{k,i+1}) \right) + \boldsymbol{D}(R_{k,LM_k}, NF_{d,k}) \end{aligned} \tag{3.2}$$

The calculation of P_k is subject to depot capacity and revenue route start time constraints.

If we let $CapN_d$ and $CapD_d$ be the capacities of all the night and day depots respectfully, then the depot capacity constraints can be modelled as:

$$\sum_{k=1}^{x} NF_{d,k} \leq CapN_d \qquad \forall\ d\ \in \{1, 2, \ldots, 7\} \tag{3.3}$$

$$\sum_{k=1}^{x} DF_{d,k} \leq CapD_d \qquad \forall\ d\ \in \{1, 2, \ldots, 7\} \tag{3.4}$$

To account for the revenue route start time constraint, a revenue route can only be added to a bus if it is possible for the bus to move from its current location to the start location of the revenue route before the start time of that revenue route. If we let $ST_{k,i}$ be the start time of revenue route $R_{k,i}$, $ET_{k,i}$ be the end time of revenue route $R_{k,i}$, $SL_{k,i}$ be the start location of revenue route $R_{k,i}$, $EL_{k,i}$ be the end location of a revenue route $R_{k,i}$, S be the speed the buss can travel, DB be the distance buffer and LB be the loading buffer, then the constraint can be modelled as:

$$ET_{k,i-1} + DB \times \frac{D(EL_{k,i-1}, SL_{k,i})}{S} + \mathrm{LB} \leq ST_{k,i} \qquad \forall\ R_{k,i}\ ; k \in \{1, 2, 3, \ldots, x\}, i \in \{2, \ldots, LA_k\} \tag{3.5}$$

The challenge in providing a solution to the internal problem is selecting the sequence of routes that each bus must operate and the depots that each bus must visit. These two things must be manipulated to create a solution that minimises the amount of busses and kilometres without violating the constraints. It must achieve this in a time of less than ten seconds so it can be useful as a model to demonstrate the impact of modifications to the fleet schedule.

3.2 Algorithm

The algorithm is the brains of the model and needs to be able to rapidly provide a good solution to the bus fleet scheduling problem.

3.2.1 Selecting the algorithm

The literature review and problem investigation provided insight on the methods available and which aspects of performance should be prioritised. Fourier-E had used the GA which works well but takes a long time to run. The investigation into modelling and the requirements of BUSCO and Fourier-E made it apparent that the algorithm needed to run fast so that it could be incorporated into the model. To cater for this speed, Fourier-E

allowed for a 20% reduction in solution quality. Being guided by Talbi (2009), the research problem was classified as a fair sized NP-hard problem, which ruled out exact methods and suggested that metaheuristics or classical heuristics were the most appropriate method. It was also identified that in general, greedy algorithms are easier to develop and generate solutions faster than iterative algorithms, but do not produce as good a solution.

Kontoravdis and Bard (1995)'s GRASP was developed to solve a very similar problem to the BUSCO bus scheduling one. The GRASP metaheuristics uses a combination of greedy logic and randomization. GRASP metaheuristics are iterative but the greedy component makes them a lot quicker than traditional iterative metaheuristics. A GRASP metaheuristics is merely an extension to a greedy algorithm that allows for randomisation. This information left three options to consider. The first was to partially develop a GRASP to the point where it is a greedy algorithm (classical heuristic). The second was to fully develop the GRASP metaheuristic. The third was to follow Fourier-E and use the Genetic Algorithm. The advantage of using the GA would be that it should produce very similar results to Fourier-E'a current algorithm. In addition, it would be easy for Fourier-E's employees to manipulate the algorithm in the future, as they are already familiar with the GA. The problem with using the GA is that it is an iterative algorithm so can take a long time to run and is stochastic, so the output varies from run to run which is not ideal. Kontoravdis and Bard (1995) provide good documentation on how to use GRASP, which made it very tempting to use. They suggest that this method is faster than most metaheuristics but it is still iterative and stochastic so its run speed and output are subject to uncertainty. The advantage of using a greedy algorithm is that the literature suggests they are easier to develop and run faster than iterative methods. The disadvantage is that the quality of solution is not as good. Since Fourier-E allowed for a relaxation of solution quality and the model requirements indicated that the run speed of the algorithm is paramount, a greedy algorithm was chosen. The greedy algorithm is the partial development of Kontoravdis and Bard (1995)'s GRASP with a few adaption's. A further advantage of using a greedy algorithm is that it is deterministic. Deterministic models instil user trust as the output is consistent. It was decided that if the greedy algorithm produced a solution in a time that was a lot faster than the 10 second constraint it would be converted into a GRASP.

The greedy components of the algorithm were broken up into functionality that minimises the number of vehicles and functionality to minimise distance. Minimising vehicles follows bin packing logic and minimising distance follows vehicle routing logic. A few algorithms were explored in the literature review that solve BBP. The one that was best suited for adaption for this project was the "Best Fit Decreasing" algorithm; however, there were two adaptions that needed to be made. The first was that instead of the routes being arranged in decreasing order, as per Kontoravdis and Bard (1995)s suggestion, they were arranged in increasing order of start time. The second was that rather than allocating the object to the bin that would be most full after the allocation, the revenue route (object) was allocated to the bus (bin) that added the least distance to the system. The algorithm that was developed as part of this project and which achieves these adaptations, has been named the "Greedy Bin" algorithm. Later in this report the Greedy Bin algorithm will be validated.

3.2.2 Algorithm logic

The logic of greedy algorithms is to make local optimal choices at every selection point. There are two important criteria when making each selection. The first is: can the revenue route be added to an existing bus instead of adding it to a new empty bus (in other words,

to minimise busses). The second is: if the revenue route can be added onto many different busses, then the revenue route must be added onto the closest bus (to minimise distance). The algorithm developed is constructive so it starts from scratch and then sequentially adds revenue routes onto busses until there are no unassigned revenue routes. The way the algorithm is coded makes it impossible to squeeze a revenue route in-between two other routes, they have to be stacked on top of each other. In order to minimise revenue routes blocking other revenue routes, the revenue routes needed to be ordered in ascending order of start time.

In addition to adding revenue routes to buses, an overnight and rest depot had to be added to each bus. The logic of this also follows greedy logic where the buses were assigned to the closest overnight and rest depots. Due to capacity constraints it was not always possible to send all the busses to the closest depot. In these cases they were sent to the second depot, but if that depot was full, they were then sent to the third depot and so on. Converting this logic into a functioning greedy algorithm was the main technical challenge of the project. The Python language was used to program the algorithm and the script can be found in Appendix A.1. The pseudo code of the algorithm is depicted in Algorithm 3 and the pseudo code of the function to identify the closest bus is presented in Algorithm 4.

Algorithm 3: The Greedy Bin Algorithm

Input : Routes $\boldsymbol{R}$, Distance Matrix $\boldsymbol{DM}$, Depot Capacities $\boldsymbol{C}$, Triger $\boldsymbol{T}$, Bus Speed $\boldsymbol{BS}$
Output: Solution $\boldsymbol{S}$, Data $\boldsymbol{D}$

```
for route in R do
    if route[Id] = T then
        for bus in busObjects do
            send to closest available day depot;
    find closest viable bus;
    if no bus found then
        create bus object;
        allocate route to bus;
        find and allocate closest available overnight depot;
    else
        allocate route to closest bus;
for bus in busObjects do
    send to overnight depot;
    extract tour data;
return S,D
```

Algorithm 4: Find Closest Viable Bus

Input : Route Start Time ***ST***, Route Start Location ***SL***,Distance Matrix ***DM***, busObjects ***BO***, Loading Time ***L***
Output: Positioning Distance ***D*** , Closest Bus ***CB***

$Avalible \leftarrow False$;
$bestDistance \leftarrow 100000$;
$b \leftarrow 0$;
for *bus in* ***BO*** **do**
    timeAvalible ← bus.Avalible;
    busLocation ← bus.Location;
    calculate distance between ***SL*** and busLocation ;
    calculate traveTime;
    if $distance \leq bestDistance$ ***and*** $timeAvalible + traveTime + \boldsymbol{L} \leq \boldsymbol{ST}$ **then**
        CB ← b;
        $bestDistance \leftarrow distance$;
        $Avalible \leftarrow True$;
    $b \leftarrow b + 1$;
D ← $bestDistance$;
if $Avalible = False$ **then**
    create bus object;
    CB ← b;
    D ← 0;
return ***D***,***CB***

A key input to the Greedy Bin algorithm is the distance matrix. Fourier-E provided a distance matrix that was generated from Google Maps. The format of this matrix is three columns with the first column being the start location, the second being the end location and the third being the road distance between the two locations. The original function that extracted the distance between locations worked through the logic seen in Algorithm 5. This method is not very efficient and resulted in the algorithm taking two minutes to execute which is too long. In order to solve this problem an alternative method was developed and resulted in the algorithm running in just over one second which is a huge improvement. To achieve this two additional programs were developed. The first program converted the original distance matrix into an NxN matrix, see Appendix A.2. The second program replaced the start and end locations in the route data with the corresponding indices of those locations in the distance matrix, see Appendix A.3. This allowed the distance to be extracted through a reference to a specific location in the distance matrix instead of looping through it, which is far more efficient.

Algorithm 5: Distance Calculator

Input : Start Location $\boldsymbol{S}$, End Location $\boldsymbol{E}$, Distance Matrix $\boldsymbol{DM}$
Output: Distance $\boldsymbol{D}$

for i *in* $\boldsymbol{D}$ **do**
  if $i[0] = \boldsymbol{S}$ ***and*** $i[1] = \boldsymbol{E}$ **then**
    $\boldsymbol{D} \leftarrow i[2]$;
    break from loop;
return ,$\boldsymbol{D}$

The basic principle that governs greedy algorithms is to make the best decision every time. On face value this logic seems ideal and in many cases it works very well, but it is prone to a problem of deteriorating selection quality as the options to choose from get exhausted. It was hoped that the effect of this tendency would not be too detrimental, but it is important to acknowledge that it could be an issue and the results of testing for it are documented later in this report.

3.2.3 Comparison and validation

The results of the algorithm and how it performed against the as-is criterion, Fourier-Es criteria and the pass criteria can be seen in Table 3.1. The results were attained with a bus speed of 40km/h, loading time of 15 min and a distance buffer of 10%. The algorithm exceeded all the pass criteria and performed particularly well in run speed. Pass criteria for busses and kilometres saved were set at 80% of those generated by Fourier-E's existing algorithm. The project's algorithm performed at 88% of Fourier-E's with regards to busses saved and 96.7% for kilometres saved. The run speed of the algorithm is 975.6 times faster than Fourier-E's. The exceptional performance in run speed makes the model user-friendly.

Table 3.1: Algorithm Results Comparison

	Busses	**Kilometres**	**Run Speed**
as-is	430	20173	na
Fourier-E	413	17988	20 min
Pass Criteria	≤ 417	≤ 18425	$\leq 10sec$
Greedy Bin Algorithm	415	18147	1.23 sec

In order to validate that the algorithm was performing as it should, a visual representation of the Duty Master was developed and the tours of 19 busses were analysed. The visual representation of the Duty Master is presented in Figure 3.1. This visual representation was particularly useful in debugging the algorithm as it made identifying errors in logic or code a lot easier.

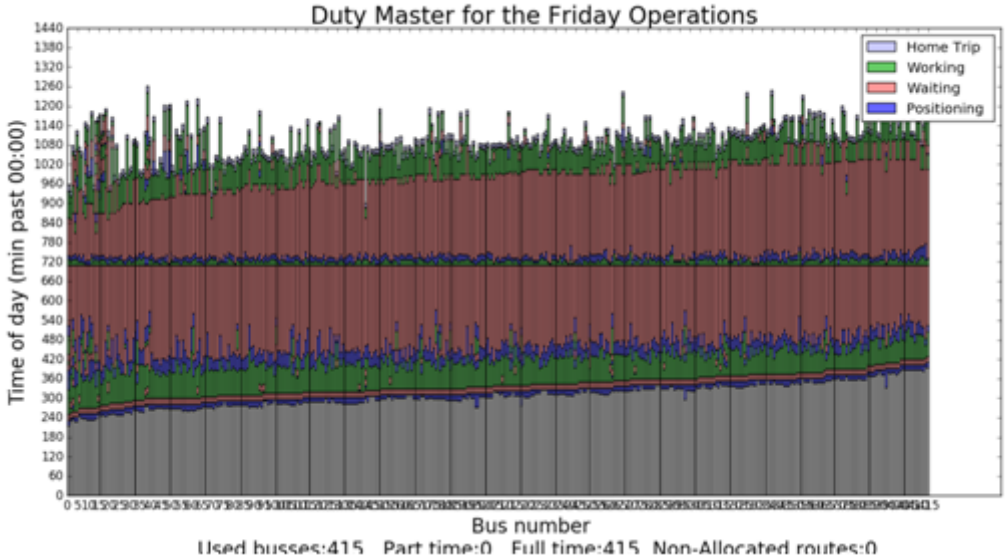

Figure 3.1: Visual representation of the Duty Master

Each bar contains the details of the time and sequence of a buses activities. Green bars represent revenue routes and by looking at the graph from left to right it is clear that the revenue routes have been ordered in ascending order of start time. The blue bars represent the positioning routes and it is these routes that the algorithm focuses on reducing distance as the revenue routes are fixed. The red bars represent the bus being idle. Before any revenue route can be executed there needs to be an idle period of at least 15 minutes to allow for the loading of customers.

The 19 busses were analysed to test that no constrains had been violated whilst the busses were executing their tours and that the algorithm was correctly calculating the positioning and transporting distances. This process involved manually testing a sample of busses and determining whether it was possible for the sequence of routes in the tour to be executed without violating any constraints and by adding up all the positioning and transporting distances, to check if they equalled the values that the algorithm provided. This process took a long time so it was impractical to test the tour of every bus. In the development stage of the algorithm some errors were detected through this method of validation. The final algorithm did not produce any errors. The algorithm was developed with an object orientated style of programming so it was easy to extract the required information. The tours that were analysed are recorded in Table 3.2.

Since the algorithm greatly exceeded all the pass criteria, there was no need to develop it further. However, the literature review revealed that a limitation of greedy algorithms is that they are prone to deteriorating selection quality as options get exhausted. To test if this tendency had materialized and if so, to what extent, a graphing function was developed that plotted the position distance that was required for each revenue route. The routes were plotted in the same order that the algorithm allocated them. The function can be found in Appendix A.4 and the output is presented in Figure 3.2

Table 3.2: Tour Analysis

Bus	Night Depot	Day Depot	Route Sequence	Positing	Transporting
1	Fordville	Pritsker	['1', '413', 'Pritsker', '544']	93.5	182.5
10	Fordville	Goldratt	['10', '461', 'Goldratt', '490']	40.7	112.8
25	Fordville	Ohno	['25', 'Ohno', '499']	45.1	132.8
50	Fordville	Ohno	['52', 'Ohno', '536', '563']	47.3	124.9
75	Fordville	Ohno	['77', 'Ohno', '556']	36.3	107.5
100	Fordville	Ohno	['104', 'Ohno', '596']	33.0	85.3
125	Fordville	Goldratt	['129', 'Goldratt', '572']	27.5	96.6
150	Fordville	Goldratt	['155', 'Goldratt', '595']	46.2	75.2
175	Fordville	Ohno	['181', 'Ohno', '680']	46.2	117.5
200	Fordville	Goldratt	['209', 'Goldratt', '636']	44.0	103.7
225	Fordville	Ohno	['237', 'Ohno', '746']	30.8	119.4
250	Fordville	Ohno	['262', 'Ohno', '772']	39.6	110.8
275	Fordville	Goldratt	['289', 'Goldratt', '717']	38.5	107.0
300	Fordville	Ohno	['314', 'Ohno', '824']	44.0	132.0
325	Fordville	Goldratt	['341', 'Goldratt', '791', '899', '969']	47.3	100.2
350	Fordville	Ohno	['366', 'Ohno', '911']	36.3	97.9
375	Nance	Fordville	['395', 'Fordville', '487']	58.3	53.1
400	Nance	Goldratt	['429', 'Goldratt', '854']	63.8	114.1
415	Nance	Goldratt	['460', 'Goldratt', '878']	51.7	87.2

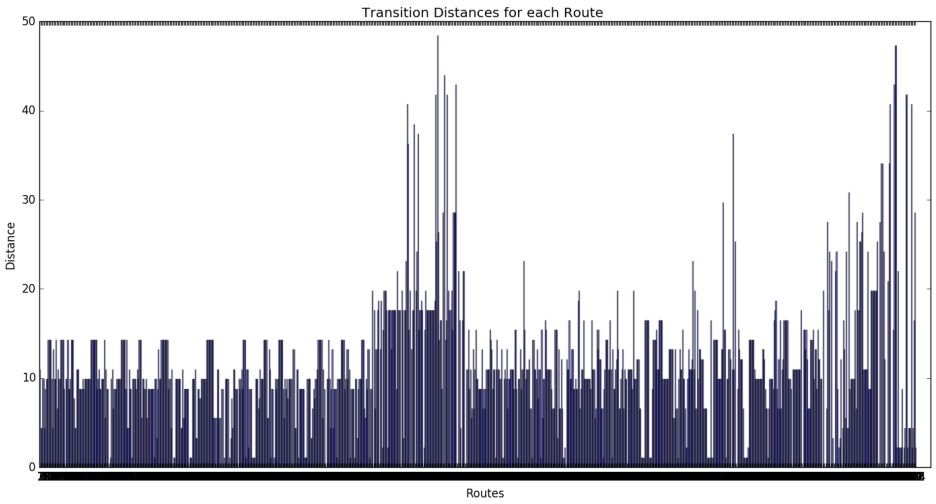

Figure 3.2: Transition Distances

Figure 3.2 clearly illustrates that the algorithm did experience a deterioration in selection quality as selection options got exhausted. There are two shifts (morning and afternoon) when revenue routes are executed with a large time gap between them. This gap resulted in the greedy algorithm being able to select from all the busses at the start of each shift; however, as time progressed into each shift, some busses became unavailable which resulted in fewer selection opportunities. The last third of revenue routes in each shift required roughly twice as many position kilometres as the first two thirds of routes. These results suggest that there is an opportunity to reduce the positioning kilometres through the application of an iterative heuristic that should reduce the total kilometres

and balance the positioning kilometres more evenly. Due to the exceptional performance in run speed, there was leeway to develop an iterative heuristic to test if this effect could be mitigated. To achieve this a GRASP was developed.

3.2.4 GRASP

Since the Greedy Bin algorithm was performing well within the run time specification and the literature suggested that given enough time iterative heuristics perform better than constructive heuristics, it was worth investigating if a GRASP metaheuristic could perform better than it. In order for the GRASP metaheuristic to be favoured it needed to perform notably better than the Greedy Bin to compensate for the inconvenience of a stochastic output and a ten second run time as specified by the upper limit of the algorithm's run-time.

Since the Greedy Bin algorithm was constructed using logic that is very similar to that of the GRASP developed by Kontoravdis and Bard (1995), it was quite simple to convert the algorithm. In order to make the conversion three things needed to be done:

- Randomise the assignment procedure of routes to busses. Kontoravdis and Bard (1995) suggest that the randomised assignment should select from the three best options and not all of the options. This creates a stochastic output which makes iteration useful.
- Add functionality to iterate over the construction of the solution.
- Add a function to save the solutions of each iteration and identify the best solution.

The output of any stochastic algorithm needs to be statistically tested as the solution varies and the degree to which it varies is important. In order to test if the GRASP metaheuristic could be a better choice for the bus fleet scheduling model, it was configured to run 500 times for ten seconds. The script that facilitated this experiment can be seen in Appendix A.5 and the results of the distances generated are seen in Figure 3.3. There was no need to display results for the number of busses employed because 99 times out of 500, 415 busses were used and the other time 414 busses were required.

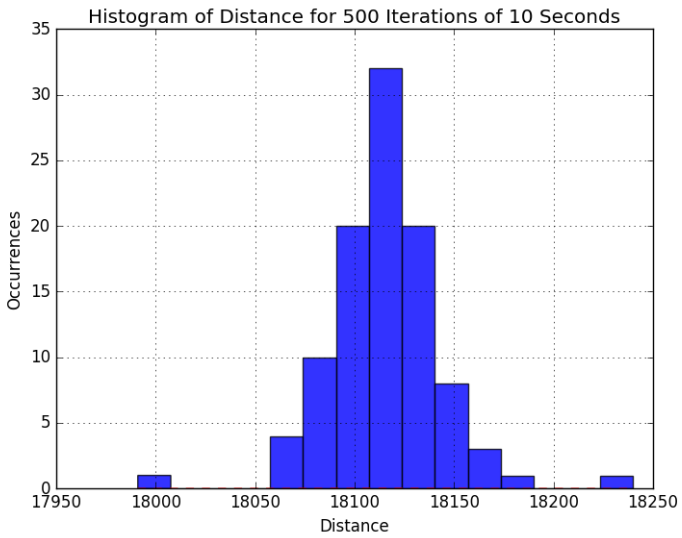

Figure 3.3: Distances of each Iteration

The statistics of this experiment for distance were:

- mean: 18112 km/d
- standard deviation: 24 km/d
- best: 17991 km/d
- worst: 18240 km/d

Compared with the Greedy Bin that required 415 busses and 18147 km/d, this experiment proved that it is possible to produce better solutions, however the gain is not sufficient to justify replacing the Greedy Bin. The mean of the GRASP is only 35 kilometres better and there is only one instance in the 500 iterations where one bus was saved .

For interest's sake, the GRASP metaheuristic was set to run overnight to test how well it could perform without the strict time constraint. With the relaxed time constraint the GRASP performed similarly to Fourier-E's GA. The results are depicted in Table 3.3.

Table 3.3: Final Algorithm Results Comparison

	Busses	**Kilometres**
as-is	430	20173
Fourier-E	413	17988
GRASP	414	17933
Greedy Bin Algorithm	415	18147

3.3 User Interface

It is impractical to provide the client with a model that they can only run using its source code because it is quite likely that the client will not have programming experience. Fourier-E specified that they wanted an online interface to interact with the algorithm. The benefit of this is that BUSCO will be able to access and use the model no matter where they are. An additional benefit is that it fits in with Fourier-E's phase two plan of opening the model up to future clients as potential clients will be able to find the model through a Google search.

In order to create a website there are three programming languages that form the basic tool kit. These languages are Hypertext Markup Language (HTML), Cascading Style Sheet (CSS) and JavaScript. In order to develop skills in these three programming languages, two websites: w3schools (2017) and stack over flow (2017), were regularly consulted. A literature review on web development was considered unnecessary as there are no complicated web development requirements of this project. Prior to discussing the functionality of the user interface, a very basic description of the three programming languages is presented.

HTML is the language that lays the scaffolding for websites. This language provides the browser with images, paragraphs and all the other bits of information that is seen on the client's browser. CSS tells the browser how the information provided by HTML should be displayed. It controls things like colour, size and positioning. JavaScript is the language that allows for more complicated functionality to be executed after a page has loaded. For example if you click on a button and the result is that the size of the text increases, this is a JavaScript instruction to the browser.

3.3.1 Interface inputs

The ability to allow the client to manipulate the inputs to the model is the functionality that should inspire the client to make changes. There are five inputs that can be changed and are explained below. The user interface developed for this project works in three steps:

1. Input values for five variables.
2. Click the solve button.
3. Display the results.

To access the user interface, please copy and paste: `http://mcgladdery.pythonanywhere.com/Simulation/` into your browser and then enter the username and password which are both "BPJ". The code for this page can be found in Appendix B.2

The five variables that the user interface allow the client to manipulate are:

1. **The speed the bus can travel**. This is used to convert the distance between locations into a travel time. This is crucial in evaluating if a bus can get in position to execute a route.
2. **The loading buffer** (minutes). It takes time for people to get on and off a bus so this needs to be accounted for. The loading buffer can also be used to make allowance for uncontrollable delays.
3. **The distance buffer** (multiplying factor). This is another mechanism to add allowances for delays. The difference between the distance buffer and the loading buffer is that the distance buffer scales with distance whilst the loading buffer is a fixed time.
4. **The capacities of night and day depots**. This allows for the effect of changing depot capacities to be evaluated.
5. **The routes to be executed**. This allows for the effect of adding new routes to be evaluated before they are implemented. This will be particularly beneficial to the client when tendering for new routes.

With the exception of the fifth variable, the other four varibles are adjusted on the main page of the simulation. The first three variables are manipulated through a select box and the capacities are changed via a number input. The inputs screen can be seen in Figure 3.4

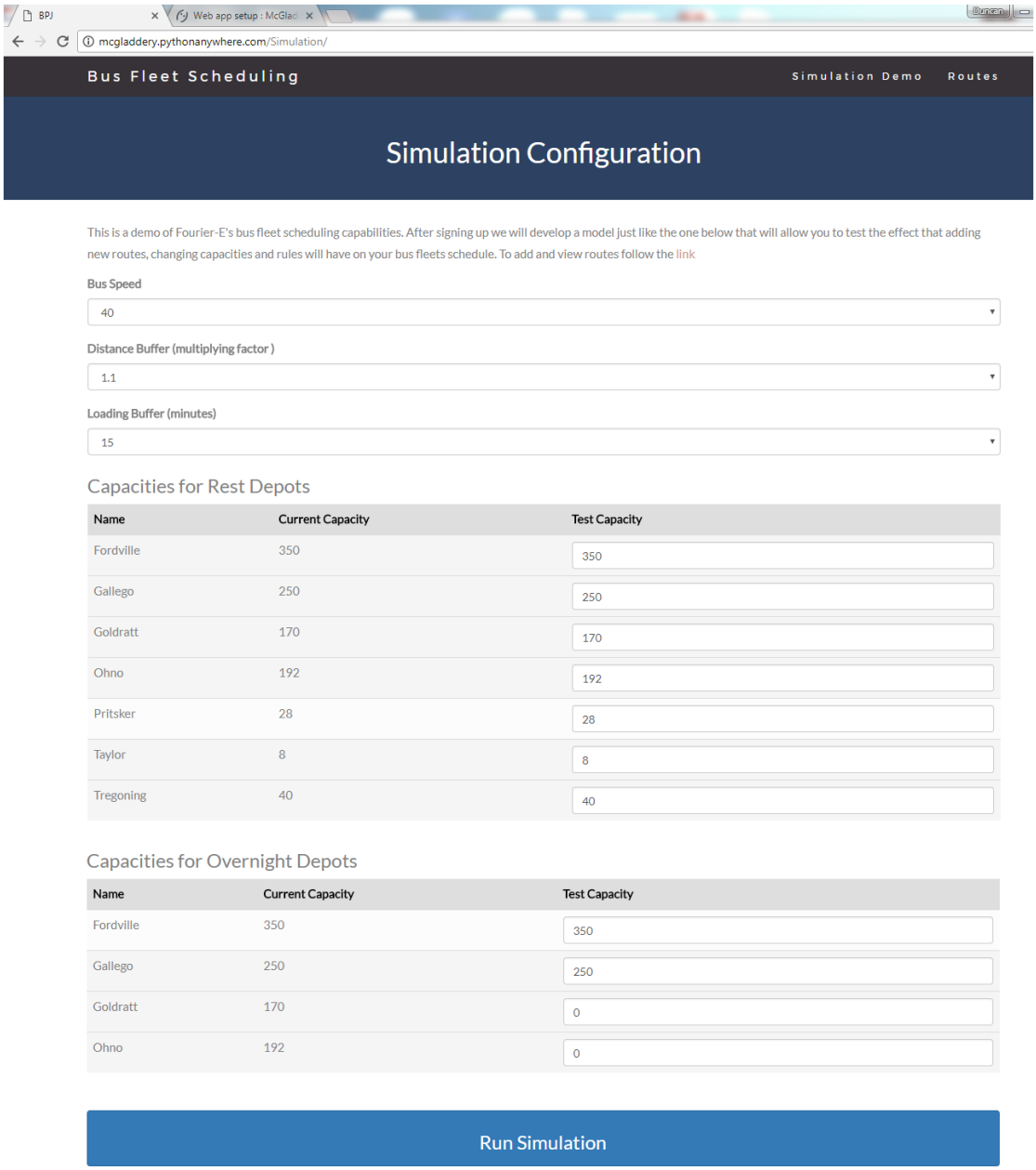

Figure 3.4: Model Input Screen

The functionality that allows users to add routes is a key component of the model. Every year new revenue routes become available so BUSCO needs to be able to add these routes into the model to determine how they will affect their operating system. The goal is for BUSCO to appreciate that the model can incorporate the new revenue routes better than they can calculate in-house.

To add this functionality, a new page was added to the website called "Routes". There are three inputs that need to be filled in to add a new route: the start time, start location and end location. After these inputs have been included the user submits this information which is then transferred to a script that inserts this new route into the file that contains all the routes. This script can be seen in Appendix A.6. After the information has been captured the page reloads and the new route can be seen in the list of routes. Figures 3.5

and 3.6 illustrate the process of adding a new hypothetical route from Alberton North to Bara that starts at 4:15 am.

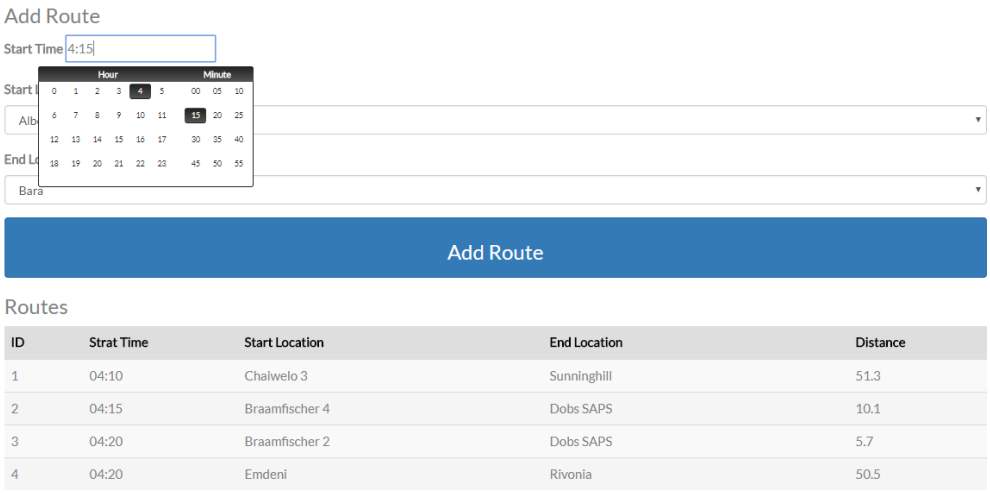

Figure 3.5: Add Route

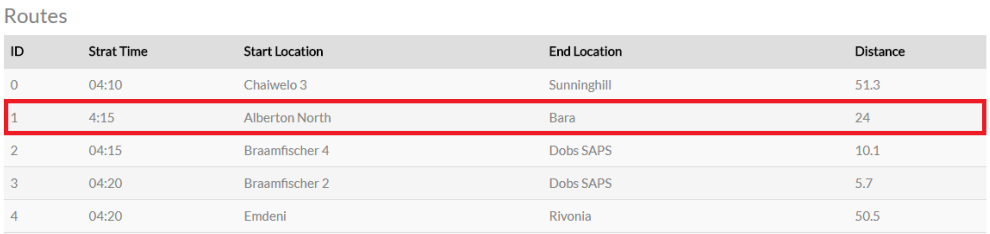

Figure 3.6: Added Route

By referring to the table in Figure 3.5 and comparing it with the table in Figure 3.6, the new route that has been added can be seen.

Many bus companies bid for the available revenue routes, so it is unlikely that BUSCO will acquire all of them. For this reason it is important to allow BUSCO to test multiple combinations of new routes. To facilitate this, it is important that it is easy to revert back to the current set of routes, so that the impact of adding new combinations can be compared in isolation. To enable this, a reset button and script have been added to the website.

3.3.2 Interface outputs

The main outputs of interest are the number of busses used and the total kilometres driven. Together, these are the main cost drivers of the bus fleet. Two other outputs are the utilisation of the bus fleet throughout the day and the utilisation of the depots. The outputs are discussed in detail in the Results chapter. Figure 3.7 illustrates how the results of the simulations are presented.

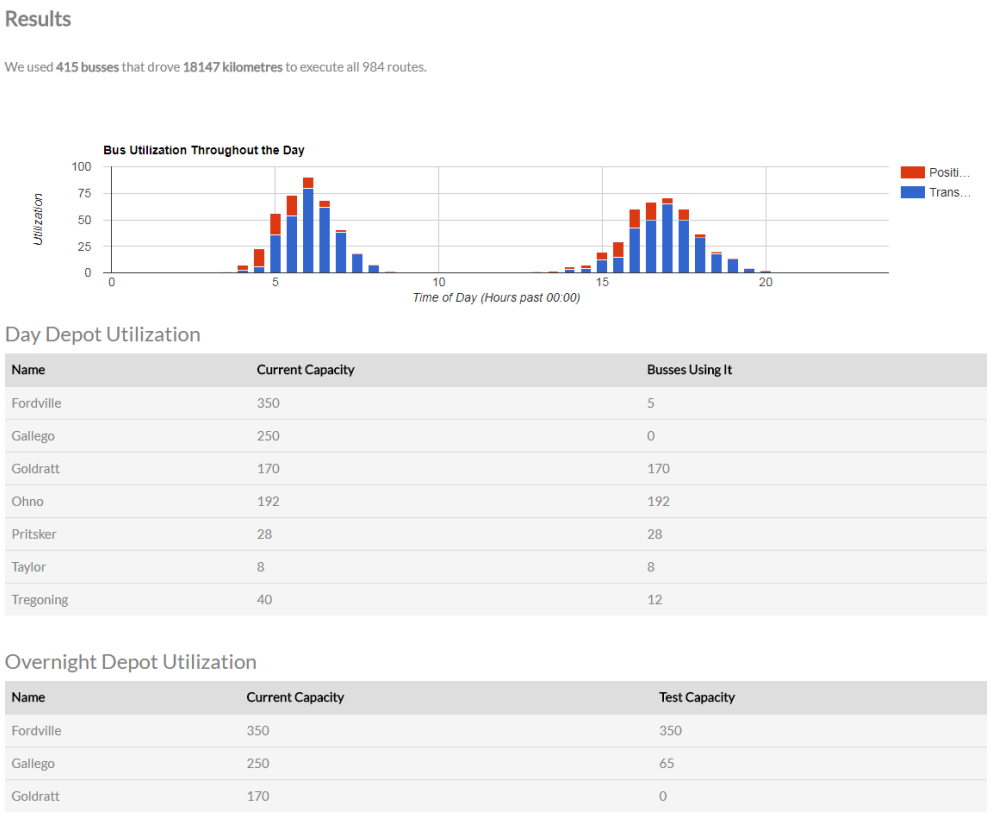

Figure 3.7: Model Output Screen

3.4 Connecting the User Interface and Algorithm

Flask is a Python web framework that facilitates the correct rendering of pages and execution of algorithms. There are a couple of alternative frameworks like web2py and dejango but there is very little difference between them. The reason Flask was chosen is that it is the most lightweight of the frameworks so is easy to get up and running. At the core of a Flask web application is a script that by convention is named "flask_app.py". This script is the link between the user's browser and the server side scripts that are broken up into: Templates, Libraries and Functions. The "flask_app.py" receives information from the client via a standard Hypertext Transfer Protocol (HTTP) or Asynchronous JavaScript And XML (AJAX) request. This script can be found in Appendix B.1. Figure 3.8 shows a flow diagram of the information transfer.

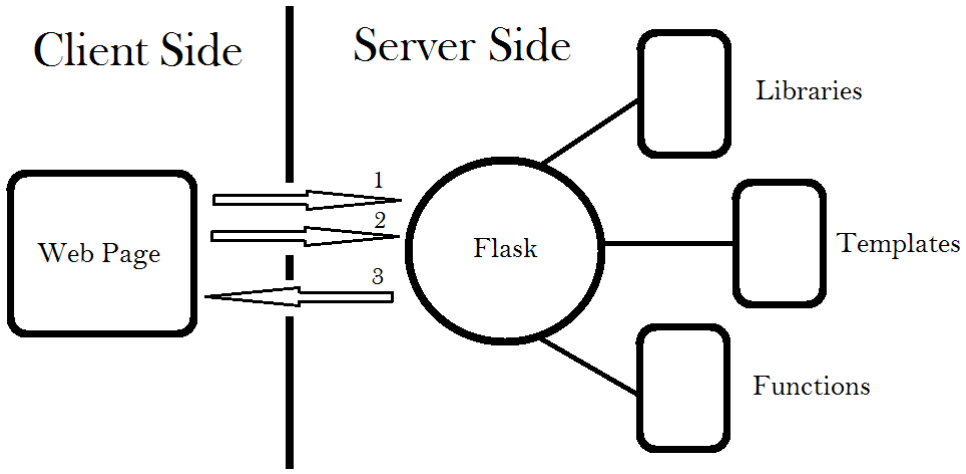

Figure 3.8: Model Flow Diagram

The templates are the scripts that contain the HTML, CSS and JavaScript and are sent to the user's browser. This information can be seen on any website by right clicking and then selecting: "View page source". The Libraries contain advanced bits of functionality that the Python programming community has made available for everyone to use. The functions contain all the advanced functionality that has personally been developed. This is where the project's algorithm is stored.

AJAX is a method that allows for information to be passed to and from the server and client without the need to reload the page. The importance of this functionality is that it is data efficient and allows the client to see an actual page whilst processing is happening in the background. The algorithm takes about 3 seconds to execute and return the data. If the alternative method was used of reloading a page via standard HTTP, the client would be looking at a white screen for a long enough period of time to think that an error might have occurred. With a bad internet connection the time taken could be even longer.

The process of how information is transferred and handled when running the BUSCO model is explained in 8 steps:

1. The client navigates to the page on the website called: Simulation. This sends a message to the server which the "flask_app.py" script then handles.

2. The "flask_app.py" script identifies that the user is not logged in, so it sends the login template back to the client.

3. The client then enters the login details and sends them back.

4. The "flask_app.py" script evaluates the login details and if they are valid it will find the Simulation template and send it back to the client.

5. The client then enters the variables into the model that is now being displayed on their browser.

6. The client then presses the Solve button which sends an AJAX request with the inputted parameters to run the simulation.

7. The "flask_app.py" script interprets this information and knows to load some libraries, run the algorithm and send the results back.

8. When the results are received, a JavaScript function displays the results and creates a graph showing how the busses have been utilised throughout the day.

3.5 Conclusion

An algorithm has been developed that is capable of solving the MDVRPPDTWIF. This algorithm has been named the: "Greedy Bin" as a result of its logic being based off greedy methods that have previously been used to solve vehicle routing and bin packing problems. The Greedy Bin algorithm exceeded all Fourier-E's specifications and performs particularly well in run speed, which is 1.23 seconds, or 975.6 times faster than Fourier-E's existing algorithm. The Greedy Bin algorithm performs at 88% of Fourier-E's with regards to busses saved and 96.7% for kilometres saved and has been validated in three ways.

An easy to use online interface has been developed with multiple web development languages. The Greedy Bin algorithm was connected to this user interface and in doing so completed the Duty Master Model. There are five variables that can be manipulated by the client, namely: bus speed, loading buffer, distance buffer and day and night depot capacities, and revenue routes. The model outputs are: number of busses needed to complete all revenue routes, positioning distance, depot allocations and fleet utilisation throughout the day.

The model has been validated and is functional and ready to use as a tool to facilitate optimisation of bus fleet scheduling.

Chapter 4

Results and Discussion

Having established the efficiency of the algorithm, the next phase of this project was to use the model as a tool to quantify the effect of making changes to the depot capacities, adding new routes and manipulating other fleet parameters.

4.1 Current Allocation vs Recommended

Table 4.1 is a comparison of the number of busses that BUSCO currently allocates to the day depots against the number of busses that the project's model recommends should be allocated to each day depot. The main difference between the two is that BUSCO allocates

Table 4.1: Current Allocation vs Recommended

Day Depot	Current	Recommended
Fordville	26	5
Gallego	10	0
Goldratt	164	170
Ohno	184	192
Pritsker	28	28
Taylor	8	8
Tregoning	10	12

a lot more busses to the Fordville and Gallego depots, whilst the model allocates more busses to Goldratt and Ohno. A limitation of this comparison is that BUSCO needs 430 busses to allocate all the routes whereas the Greedy Bin algorithm reduces the number of busses to 415. Whilst acknowledging this limitation, a soft estimate of the potential gain in efficiency achieved by allocating the busses according to the model's recommendation can be calculated through a simulation. By constraining the capacities of the day depots to that of BUSCO's current allocation, the Greedy Bin algorithm is forced to send roughly the same amount of busses as BUSCO to each day depot. When the simulation was run under these capacity constraints it demonstrated that BUSCO's less than optimal allocation is adding an extra 472 kilometres per day onto the total positioning distance. If we assume that there are four weeks in a month and the daily routes are repeated five times a week, then an additional 9440 km are driven every month. BUSCO assigns a fuel and maintenance cost of R13 per kilometre so the total cost of the current allocation can be estimated as R122 720 per month more than it would be if the model's output was implemented.

The exact simulation can be run on the night depots. Currently BUSCO allocate 300 busses to Fordville and 130 busses to Gallego. When the experiment was run for the night depots, an additional 747 kilometres were added, which amounts to a wastage of R194 220 per month. These simulations suggest that changing the capacity and location of the depots is a big cost driver.

4.2 Simulations with Changing Day Depot Capacities

The as-is configuration uses a bus speed of 40 km/h, distance buffer of 1.1, loading buffer of 15 minutes and sets the day depots capacities to those provided by BUSCO. The simulation in this section used the as-is configuration for everything except depot capacity, which represents the simulation variable. The current capacities of the day depots and the number of busses that the algorithm recommends should use them is shown in the screen shot of the algorithm's output (Figure 4.1). In computing the simulation, the algorithm was able to allocate all 984 revenue routes to 415 busses that drove 18147 km.

Day Depot Utilization

Name	Current Capacity	Busses Using It
Fordville	350	5
Gallego	250	0
Goldratt	170	170
Ohno	192	192
Pritsker	28	28
Taylor	8	8
Tregoning	40	12

Figure 4.1: Recommended Day Depot Allocation

When comparing recommended utilisation versus capacity of the day depots, it became apparent that there were a number of inefficiencies in BUSCO's current Duty Master. These are discussed in the next sections of the report.

4.2.1 Depot underutilization

Fordville is severally underutilized during the day. It has a capacity for 350 busses but only 5 busses are currently allocated to it. Fordville has the largest capacity because it is also being used as one of the two overnight depots. It is currently kept open 24/7 but it might serve BUSCO better by closing it during the day. The salaries and operating expenses of running Fordville during the day are not available, but the model allows for the expense of closing Fordville depot during the day to be calculated. By setting the capacity of Fordville to zero during the day, the model outputs that 415 busses are required and 18206 kilometres driven. This is 59 kilometres more per day than the as-is scenario, which amounts to an extra 1180 kilometres driven per month. With a fuel and maintenance cost of R13 per kilometre, closing Fordville during the day would add an expense of R15 340 per month. The cost of running Fordville has not been provided by BUSCO, but it is likely to be more than R15 340 per month as many staff are employed. If this is the case, then BUSCO should consider closing the Fordville depot during the day. Similarly, the model recommends that no busses should be sent during the day to the Gallego depot, which is the other overnight depot, so this is another opportunity for the client to reduce expenditure by cutting the day staff crew.

Tregoning is the last depot that has a discrepancy between its capacity (40) and the amount of vehicles assigned to it (12) by the project's model. This suggests that Tregoning might not be well located for functioning as a day depot. To evaluate this assumption, the capacity of the Tregoning depot was set to zero and the output of the model evaluated. The effect of rerouting the 12 busses that are currently assigned to it is substantial. Closing Tregoning as a day depot adds an additional 624 kilometres per day which amounts to R162 240 additional expenditure every month. It seems very unlikely that the cost of operating the depot will exceed R162 240 per month, so despite being utilised at only 33.3% of its capacity, it should not be closed.

4.2.2 The potential for increasing depot capacity

The model indicates that Goldratt, Ohno, Pritsker and Taylor depots are all utilized to capacity. This suggests that these depots are in favourable positions and that by increasing their capacity a saving in distance driven could result. To evaluate this, the capacity of all four depots was individually increased to the ideal capacity. The ideal capacity is established by setting the capacity of the depot to a very large number which essentially eliminates the constraints. The number of busses that the algorithm allocates to the depots is considered the ideal capacity for each depot. The results of these simulations and the monthly financial implications of changing the day depot capacities are shown in Table 4.2. The monthly financial implications are calculated by multiplying the increase or decrease in kilometres driven every month by R13.

Table 4.2: Capacity Increase Simulation

Depot	**Current**	**Ideal**	**Effect on Rand**
Goldratt	170	173	(7280)
Ohno	192	192	0
Pritsker	28	31	4160
Taylor	8	12	8320

From the above table it is apparent that it would be beneficial to increase the current capacity of the Pritsker or Taylor depot. It is unexpected that there is a negative effect on cost when relaxing the capacity constraint of Goldratt. This is an effect of the greedy logic where the best selection is made at the present moment without regard for what the effects will be later on. There would have been a benefit to send an extra three busses to Goldratt at the end of the morning shift, but these three busses would have had to travel further to get into position for their afternoon routes. Running the simulation for Ohno suggested that the current capacity is also the ideal capacity for that depot.

One final simulation was run where all the day depot capacities were relaxed at once. The model recommended that the capacities of Goldratt and Pritsker be increased by three and that Taylor be increased by four. This resulted in a saving of 82 kilometres per day, which amounts to R21 320 per month.

4.3 Simulations with Changing Overnight Depot Capacities

Goldratt and Ohno depots are currently just being used as day depots but since they already have a large capacity and are favoured over Fordville and Gallego as day depots, it was hypothesised that converting these two depots to overnight depots could reduce the

total kilometres driven. To cater for this, Goldratt and Ohno were added onto the model as overnight depots. When the model was run with these depots included it was surprising that at nighttime, the two depots were not at all favoured like they were during the day. The model's output are depicted in the figure below.

Overnight Depot Utilization

Name	Current Capacity	Test Capacity
Fordville	350	350
Gallego	250	65
Goldratt	170	0
Ohno	192	0

Figure 4.2: Recommended Overnight Depot Utilisation

In total, only four busses were assigned to the Goldratt and Ohno depots and there was no saving in kilometres, so it does not make sense to convert these two depots into overnight depots. The reason for this result has not been investigated but it is suspected that it is a consequence of Goldratt and Ohno being located in the city centre and Fordville and Gallego being located in the urban periphery. The general commuting pattern of bus passengers is to travel from the outskirts of the city into its centre in the morning and then back again in the afternoon. If Goldratt and Ohno are used as overnight depots, then the busses will have to travel all the way to and from the city centre to the periphery each day just to get into position, whereas having the busses overnight at Fordville and Gallego, means they are positioned close to the start and end of the daily commuting route.

Fordville's depot is being utilised to capacity in the evening. This suggests that compared to Gallego it is in a more favourable position. When the capacities of the day depots were relaxed the model only made minor changes to the suggested capacities. However, this was not the case with the overnight depots. The model assigned 410 busses to Fordville and only five to Gallego. As it seems hard to justify keeping a depot open overnight for only five busses, a second simulation was run with the capacity of Gallego set to zero. By allocating all the busses to Fordville, a saving of 879 kilometres every day resulted, which amounts to R228 540 per month. This is a significant saving. The cost of increasing the capacity of Fordville to cater for an additional 65 busses can now be evaluated against its associated saving. It might not be possible to increase the capacity of Fordville, but this simulation clearly indicates that there could be substantial financial gains in doing so.

4.4 Simulations with Adding Routes

Fourier-E provided four routes to be used to test the effect of adding new revenue routes. The route details are depicted in Table 4.3. Adding these routes to the model resulted in an additional 72 kilometres per day being driven and the addition of one bus. This information is valuable as it will enable BUSCO to compare the additional revenue versus the expected costs of adding the new routes. An additional output of the model is a bar chart of the bus fleet's utilization throughout the day. This chart can be seen in Figure 4.3 and is used to identify which periods of the day are underutilized and which period is the constraint. This information is useful in predicting if new routes can be added without adding extra busses. It is clear that the middle of the morning period (6:00 to 6:30) is the most busy with a fleet utilization of 90%. One of the four test routes is in this period of the day so it was worth checking the effect of excluding that route but incorporating the other three. The result of adding the three routes was an additional 144 kilometres

Table 4.3: Capacity Increase Simulation

Start Time	From	To	Distance (km)
04:15	Angus	Emdeni	35
06:25	Waterfall	Rivonia	9
15:30	Sandton	Esso	14
18:00	Kyalami	Kosmosdal	18

driven but no extra busses were required. It may come as a surprise that adding three routes resulted in more kilometres than adding four routes. The reason for this is that the additional bus that was required when adding four routes provided the Greedy Bin algorithm with more options to allocate routes to busses. If the revenue that each route generates was provided, then the financial impact of adding the routes would be easy to calculate and in doing so would empower Fourier-E to make an informed decision. In the test case, the extra revenue from the 4th route would have to exceed the extra expense of operating an additional bus. If the extra revenue does not exceed the expense then BUSCO should only tender for the three routes and not the one that is in the constrained mid-morning period.

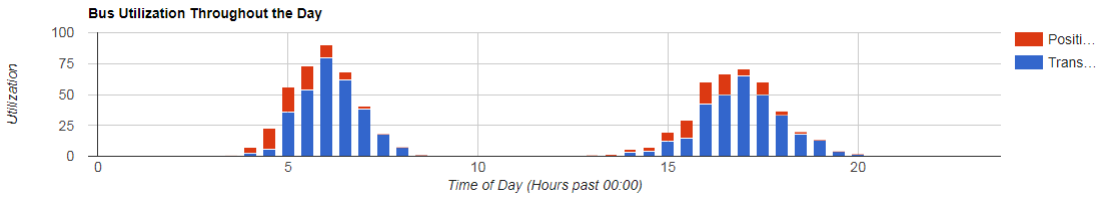

Figure 4.3: Bus Utilisation Throughout the Day

4.5 Simulation for Potential Performance and Implementation

Discussions between Fourier-E and BUSCO have indicated that there is opposition from the unions with regards to the practicality of implementing the proposed Duty Master that is based on Fourier-E's original algorithm. The unions are skeptical that it is possible to execute all 984 routes with just 413 busses instead of the 430 busses currently in operation (refer to Table 3.1). Since a primary function of the unions is to protect the drivers from job loss, they understandably are not supportive of a Duty Master that requires 17 less busses. In order to convince the unions that an extreme approach has not been taken, the new model that has been developed for this project can be used to demonstrate the effect that adding buffers and changing the speed of the busses will have.

4.5.1 Increasing bus speed

The first simulation illustrates the effect of increasing the bus speed from 40 km/h to 50 km/h while maintaining a 15 minute loading buffer. These parameters do not allow for many delays but are not unrealistic. This configuration results in 396 busses being used and 18215 kilometres travelled. This simulation can provide BUSCO with some negotiating power as it is evident that there is a feasible solution that is considerably better in terms

of buses required (396 instead of 413) than the current one being proposed.

4.5.2 Manipulating buffer parameters

By manipulating the loading and distance buffers, the model can be used to demonstrate to BUSCO the effects of those changes on the Duty Master in terms of total distance travelled and number of busses required to implement those changes. Table 4.4 shows the effect of changing the loading buffer while maintaining the bus speed at 40 km/h and the distance buffer set to 1. Table 4.5 shows the effect of changing the distance buffer while maintaining the bus speed at 40 km/h and setting the loading buffer to zero.

Table 4.4: Varying Loading Buffer

Loading Buffer	**Busses**	**Kilometres**
0	400	18034
5	406	18047
10	408	18198
15	412	18156
20	415	18150
25	422	18183
30	427	18180

Table 4.5: Varying Distance Buffer

Distance Buffer	**Busses**	**Kilometres**
1	400	18034
1.05	401	18008
1.1	405	17957
1.15	405	17984
1.2	406	17976
1.3	409	18006

On an impractical note, all routes can be allocated to 252 busses if there are no buffers and the busses maintain an average speed of 100km/h.

The algorithm and associated model that have been developed provide a rapid and easy to use tool for demonstrating how the manipulation of various components of a bus fleet schedule can result in greater efficiencies and in doing so lead to significant savings in operating expenditure.

Chapter 5

Conclusion

Prior to commencing this project, Fourier-E, an industrial engineering consultancy company, had developed an algorithm to optimise BUSCO's Duty Master. A limitation of their algorithm was its slow computational time of approximately 20 minutes. Through studying the work of Kontoravdis and Bard (1995) and Talbi (2009), an algorithm was developed that exceeded all Fourier-E's specifications and performed particularly well in run speed, which at 1.23 seconds, is 975.6 times faster than Fourier-E's algorithm. This algorithm was named the "Greedy Bin" as its logic is based on greedy methods to solve vehicle routing and bin packing problems. The algorithm performed at 88% of Fourier-E's with regards to busses saved and 96.7% for kilometres saved.

The next requirement was to covert the algorithm into a model by developing an online user interface that incorporated the algorithm. Through using a variety of web development programming languages the online interface was created. The Python based Flask framework allowed for the Greedy Bin algorithm to be incorporated into the model.

The model allows the client, BUSCO, to see the benefits of making changes to their operations which can then be used to generate a new Duty Master which can be compared with their existing fleet schedule. Five variables that can be manipulated by the client were added to the user-interface, namely: bus speed, loading buffer, distance buffer , revenue routes and day and night depot capacities. The model outputs are: the number of busses needed to complete all revenue routes, positioning distance, depot allocations and fleet utilization throughout the day. The model is easy to use and rapidly generates it's output.

The model has been used to identify short falls and opportunities in BUSCO's current operations. The most apparent short fall is that the overnight depot Gallego is in a very poor location. The manipulation of the overnight depot capacities revealed that by closing down Gallego and increasing the capacity of Fordville, a saving of R228 540 per month in fuel and maintenance, plus the cost associated with running Gallego, could be made. This potential saving warrants an investigation into the practicality of increasing the capacity of Fordville and/or identifying a new location for a second overnight depot.

At a more general level, manipulation of the model using various what-if scenarios has demonstrated its ability to save costs through the identification of more efficient depot allocations, changing the capacity of existing favourably located depots, and through enhancing scheduling performance by increasing bus speeds and reducing buffer times. In the highly competitive world of tendering, the model enables clients to determine the costs and benefits of new revenue routes prior to deciding whether or not to bid for them.

This project identified potential savings in terms of busses needed or kilometres reduced and also demonstrated the financial implications of changing depot allocations and

capaities.

A suggestion for future research is to add another variant of the VRP, namely the Heterogeneous Fleet. The busses in the Sowetan operations have the same capacity but this is not the case in some of the other regions that have normal busses and bi-articulated busses. When there are multiple types of vehicles in the fleet the problem is considered to be a Heterogeneous Fleet Vehicle Routing Problem (HVRP). Incorporating this variant will add a lot of complexity but expand the area to which the model can be applied.

The successful development of the "Greedy Bin" algorithm as a method to solve the Mulit-Depot Vehicle Routing Problem with Pickup and Delivery, Time Windows and Intermediate Facilities (MDVRPPDTWIF) and the programming of its associated user interface, has created an easy to use online tool that can speedily generate a Duty Master. As such, this project has demonstrated the potential of the model to rapidly facilitate the optimisation of the scheduling of a large bus fleet.

To test the model please visit the website: `http://mcgladdery.pythonanywhere.com/Simulation/`. The password and username for both is: "BPJ".

Bibliography

Aaronson, S. (2006). *Scott Aaronson Blog, Available at https://jsprit.github.io/.* [Online; accessed 04-September-2017 at https://jsprit.github.io/].

Box, G. E., Draper, N. R., et al. (1987). *Empirical model-building and response surfaces*, volume 424. Wiley New York.

Braekers, K., Ramaekers, K., and Van Nieuwenhuyse, I. (2016). The vehicle routing problem: State of the art classification and review. *Computers & Industrial Engineering*, 99:300–313.

BUSCO (2017). *BUSCO Alias for the bus company.* Alias for the bus company [Online; accessed 04-September-2017 at http://anonymous.co.za/].

Claymathn (2017). *Claymathn.* [Online; accessed 04-September-2017 at http://www.claymath.org/millennium-problems/rules-millennium-prizes].

JSPRIT (2017). *jsprit is a java based, open source toolkit for solving rich traveling salesman (TSP) and vehicle routing problems (VRP).* [Online; accessed 04-September-2017 at https://jsprit.github.io/].

Kontoravdis, G. and Bard, J. F. (1995). A grasp for the vehicle routing problem with time windows. *ORSA journal on Computing*, 7(1):10–23.

Manson, N. (2006). Is operations research really research? *ORiON: The Journal of ORSSA*, 22(2):155–180.

Martello, S. and Toth, P. (1990). *Knapsack problems: algorithms and computer implementations.* Wiley-Interscience series in discrete mathematics and optimization. J. Wiley & Sons.

Pidd, M. (2010). Why modelling and model use matter. *Journal of the Operational Research Society*, 61(1):14–24.

stack over flow (2017). *stackoverflow worlds largest developer community.* [Online; accessed 04-September-2017 at https://www.stackoverflows.com/].

Talbi, E.-G. (2009). *Metaheuristics: from design to implementation*, volume 74. John Wiley & Sons.

w3schools (2017). *W3Schools worlds largest developer site.* [Online; accessed 04-September-2017 at https://www.w3schools.com/].

Appendix A

Python

A.1 Algorithm

```
"""This algoruthm uses a distance matrix"""
import csv
import Analytics


def Run(speed,capacities,nightCap,LB,DB):
    global LoadingTime,buffer
    LoadingTime = LB
    buffer=DB
    #-------------------------------------------------------------------------
    #Global paramiters
    global DobsNightCap,nameDay,locDay,busObjects,cntDay,capDay,cntDobsNight,\
    cntNanceNight,cntNight,capNight,speedLimit,distanceMatrix\
    ,transitionDistances
    #-------------------------------------------------------------------------
    #varibles

    sendToDayDepot = '480'
    DobsNightCap = 350
    cntDobsNight = 0
    cntNanceNight = 0

    nameDay = ['Dobs','Nance','Kya Sand','Linbro Park','Strydom Park','Steeldale',\
               'Selby']
    locDay = ['33','80', '65','69','108','105','98']
    capDay = [350,250,170,192,28,8,40]
    cntDay = [0]*len(capDay)
    capNight = [0,0,0,0]
    cntNight = [0]*len(capNight)

    #Data extraction & storage
    Routes = []
    busObjects = []
    distanceMatrix = []
    transitionDistances = []
    route_csv = open('Friday_Index.csv','r')
    reader = csv.reader(route_csv)
    cntRoutes = 0
    for i in reader:
        Routes.append(i)
        cntRoutes +=1
    route_csv.close()

    matrix = open('newDistanceMatrix.csv','r')
    reader = csv.reader(matrix)
    for i in reader:
        distanceMatrix.append(i)
    matrix.close()

    capacities = [float(x) for x in capacities]
    nightCap = [float(x) for x in nightCap]

    capDay = capacities
    capNight = nightCap

    speedLimit = speed
    #-------------------------------------------------------------------------


    #-------------------------------------------------------------------------
    #Algorithm Main
    for routeData in Routes:

        if routeData[0] == sendToDayDepot:
            for b in busObjects:
                b.Add_Day_Depot()
```

```
        bus,pD=Closest_Bus(routeData[1],routeData[2])

        if busObjects[bus].Routes == []:     #Add BaseDepot
            pD=busObjects[bus].Add_AM_Depot(routeData[2],routeData[1])

        busObjects[bus].Add_Route(routeData[0],pD,float(routeData[4]),routeData[1]\
        ,routeData[3])

    #Data + Home trip
    PositioningDistance = 0
    for bus in busObjects:
        bus.Add_PM_Depot()
        PositioningDistance = PositioningDistance+bus.PositioningDistance

    data = Analytics.Utilization(busObjects)

    return len(busObjects),round(PositioningDistance,2)/buffer,cntRoutes,cntDay,cntNight,data
#-------------------------------------------------------------------------------
#Classes
class Bus():

    def __init__(self):
        self.Routes =[]
        self.Tour = []
        self.PositioningDistance = 0
        self.RouteDistance = 0
        self.avalible = 0
        self.loc=0
        self.BaseDepot = ''
        self.StartTime = 0
        self.EndTime = 0

    def Add_Route(self,routeId,pDistance,tDistance,startTime,endLoc):
        global LoadingTime
        startTime = Normalized_Time(startTime)
        self.Routes.append(routeId)
        self.Tour.append(['p',startTime-LoadingTime-60*pDistance/(speedLimit*1.0),\
                          startTime-LoadingTime])
        self.Tour.append(['t', startTime,startTime+ 60*tDistance/(speedLimit*1.0)])
        self.avalible = startTime + 60*tDistance/(speedLimit*1.0)
        self.loc= endLoc
        self.PositioningDistance = self.PositioningDistance + pDistance
        self.RouteDistance += tDistance
        transitionDistances.append(round(pDistance,2))

    def Add_AM_Depot(self,startLoc,startTime):
        startTime = Normalized_Time(startTime)
        nightOptions = []
        nightOptionsId = []
        for i in range(4):
            if cntNight[i] < capNight[i]:
                dist = Distance(locDay[i],startLoc)
                nightOptions.append(dist)
                nightOptionsId.append(i)

        ind = nightOptions.index(min(nightOptions))
        bestDist = nightOptions[ind]
        ind2 = nightOptionsId[ind]
        cntNight[ind2] +=1
        self.BaseDepot = nameDay[ind2]
        self.loc = locDay[ind2]
        self.StartTime= startTime

        return bestDist

    def Add_Day_Depot(self):
        bestDist = 1000
        for i in range(len(capDay)):
            if cntDay[i] < capDay[i]:
                dist = Distance(self.loc,locDay[i])
                if dist < bestDist:
                    bestDist = dist
                    index = i
        self.Routes.append(nameDay[index])
        self.PositioningDistance = self.PositioningDistance + bestDist
        self.loc=locDay[index]
        self.RestDepot = nameDay[index]
        cntDay[index] += 1


    def Add_PM_Depot(self):
        if self.BaseDepot == 'Dobs':
            d=Distance(self.loc, '33')
            self.PositioningDistance = self.PositioningDistance + d
            self.loc= '33'
            self.EndTime = self.avalible + 60*d/(speedLimit*1.0)
        elif self.BaseDepot == 'Nance':
            d=Distance(self.loc, '80')
            self.PositioningDistance = self.PositioningDistance + d
            self.loc= '80'
            self.EndTime = self.avalible + 60*d/(speedLimit*1.0)
        elif self.BaseDepot == 'Kya_Sand':
            d=Distance(self.loc, '65')
            self.PositioningDistance = self.PositioningDistance + d
```

```
            self.loc= '65'
            self.EndTime = self.avalible + 60*d/(speedLimit*1.0)
        elif self.BaseDepot == 'Linbro_Park':
            d=Distance(self.loc, '69')
            self.PositioningDistance = self.PositioningDistance + d
            self.loc= '69'
            self.EndTime = self.avalible + 60*d/(speedLimit*1.0)


#-----------------------------------------------------------------------
#Functions
def Closest_Bus(startTime,startLoc):
    Avalible = False
    startTime = Normalized_Time(startTime)
    bus = 1000
    travelDistance= 1000
    cnt = 0
    for i in busObjects:

        distance= Distance(i.loc,startLoc)
        pTime = 60*distance/(speedLimit*1.0)
        if distance < travelDistance and i.avalible+pTime +LoadingTime\
        <startTime:
            Avalible = True
            bus = cnt
            travelDistance = distance

        cnt +=1

    if Avalible == False:
        busObjects.append(Bus())
        bus = cnt
        travelDistance = 0
    return bus,travelDistance


def Distance(loc1, loc2):

    distance= distanceMatrix[int(loc1)][int(loc2)]

    return float(distance)*buffer


def Normalized_Time(time):
    time = time.split(":")
    normalized_time = int(time[0])*60+int(time[1])
    return normalized_time

a,b,c,d,e=Run(40,[350,250,170,192,28,8,40],[350,250,170,192],15,1.1)
print(a,b,c,d,e)
#Analytics.Total_Working_Hours(busObjects)
#Analytics.Routes_Driven_Data(busObjects)
#Analytics.Positioning_Distances(transitionDistances)
#Analytics.Utilization(busObjects)

'''for i in [0,9,24,49,74,99,124,149,174,199,224,249,274,299,324,349,374,399,414]:
    print(str(i+1) +' & '+ busObjects[i].BaseDepot +' & '+ busObjects[i].RestDepot\
    +' & '+ str(busObjects[i].Routes)+' & '+ str(round(busObjects[i].PositioningDistance,2))\
    +' & '+ str(round(busObjects[i].RouteDistance,2)) + "\\")'''
```

A.2 Distance Matrix Formatter

```
import csv
#-----------------------------------------------------------------------
#Extract Old Distance Matrix Data
oldDistanceMatrix = []
matrix = open('DistanceMatrix.csv','r')
reader = csv.reader(matrix)
for i in reader:
    oldDistanceMatrix.append(i)
matrix.close()

#-----------------------------------------------------------------------
#Get all Locations
Locations = []
for i in oldDistanceMatrix:
    if i[0] not in Locations:
        Locations.append(i[0])
Locations.pop(0)
#-----------------------------------------------------------------------
#Distance Finder
def Distance(loc1, loc2):
    found = False
    for i in oldDistanceMatrix:

        if i[0]== loc1 and i[1]==loc2:
            distance = i[2]
            found = True
            break
    if found ==False:
        print(loc1,loc2)

    return distance
```

```
#-------------------------------------------------------------------------------
#Create New Matrix
newDistanceMatrix=[]
cnt =0
outputFile = open('newDistanceMatrix.csv', 'w')
for i in Locations:
    if cnt ==0:
        outputFile.write('NULL,')
        for k in Locations:
            outputFile.write(k+',')
        outputFile.write('\n')

    rowData=[]
    cnt2=0
    for j in Locations:
        if cnt2==0:
            outputFile.write(i+',')
        d= Distance(i,j)
        rowData.append(d)
        outputFile.write(d+',')
        cnt2 +=1
    newDistanceMatrix.append(rowData)
    outputFile.write('\n')
    cnt+=1
print(newDistanceMatrix)
outputFile.close()
#-------------------------------------------------------------------------------
```

A.3 Route Formatter

```
import csv
#-------------------------------------------------------------------------------
#Extract Distance Matrix Data
DistanceMatrix = []
matrix = open('newDistanceMatrix.csv','r')
reader = csv.reader(matrix)
for i in reader:
    DistanceMatrix.append(i[0])
matrix.close()

#Extract Routes
Routes=[]
route_csv = open('Friday_Name.csv','r')
reader = csv.reader(route_csv)
cntRoutes = 0
for i in reader:
    Routes.append(i)
    cntRoutes +=1
route_csv.close()

#-------------------------------------------------------------------------------
#Create Index Routes

cnt =0
outputFile = open('Friday_Index.csv', 'w')
for i in Routes:
    indLoc = 0
    for loc in DistanceMatrix:
        if loc == i[2]:
            indStart = indLoc
        if loc == i[3]:
            indEnd = indLoc
        indLoc +=1

    outputFile.write(i[0]+','+i[1]+','+str(indStart)+','+str(indEnd)+','+i[4]+'\n')

    cnt+=1

outputFile.close()
```

A.4 Functions For Analysis

```
import matplotlib.pyplot as plt; plt.rcdefaults()
import numpy as np
import matplotlib.pyplot as plt
#Total Hours Bar
def Total_Working_Hours(busObjects):
    performance = [0]*49
    hoursData = []
    objects = [x for x in np.arange(1,17,0.33)]
    print(len(performance),len(objects))
    for bus in busObjects:
        hours = (bus.EndTime - bus.StartTime)/60.0
        hoursData.append(hours)
        hours = str(hours)
        hours = hours.split('.')
        indPerformance= int(hours[0])*3 -1
        if int(hours[1][0]) < 3.3:
            indPerformance +=1
```

```
            elif int(hours[1][0]) < 6.6:
                indPerformance +=2
            elif int(hours[1][0]) < 9.99:
                indPerformance +=3
            performance[indPerformance] +=1

        '''tuple(objects)
        y_pos = np.arange(len(objects))
        plt.bar(y_pos, performance, align='center', alpha=0.5)
        plt.xticks(y_pos, objects)
        plt.ylabel('Number of Busses')
        plt.title('Spread of bus utilisation by hours worked ')

        plt.show()'''
        x=len(hoursData)
        hoursData=sorted(hoursData)
        #print(hoursData)
        y_pos = np.arange(x)
        plt.bar(y_pos, hoursData, alpha=0.5)
        plt.xticks(y_pos, objects)
        plt.ylabel('Hours')
        plt.title('Spread_of_bus_utilisation_by_hours_worked_')

        plt.show()

#-------------------------------------------------------------------------------
#Total Hours Driving
def Total_Driving_Hours(busObjects):
    performance = [0]*24
    objects = [x for x in np.arange(1,13,0.5)]
    print(len(performance),len(objects))
    for bus in busObjects:
        hours = (bus.PositioningDistance + bus.RouteDistance)/60.0
        indPerformance= int(hours-0.5)*2
        if hours %1 <.5:
            indPerformance +=1
        performance[indPerformance] +=1

    tuple(objects)
    y_pos = np.arange(len(objects))
    plt.bar(y_pos, performance,  alpha=0.5)
    plt.xticks(y_pos, objects)
    plt.ylabel('Number_of_Busses')
    plt.title('Spread_of_bus_utilisation_by_hours_driven_')

    plt.show()

#-------------------------------------------------------------------------------
#Routes driven Data
def Routes_Driven_Data(busObjects):
    routeBins=[0]*15
    objects = [x for x in np.arange(1,16,1)]
    avg = 0
    for bus in busObjects:
        indBin= len(bus.Routes)-2
        routeBins[indBin]+=1
        avg+= indBin +1
    avg =round( avg/len(busObjects),2)
    print(avg)
    tuple(objects)
    y_pos = np.arange(len(objects))
    plt.bar(y_pos, routeBins,  alpha=0.5)
    plt.xticks(y_pos, objects)
    plt.ylabel('Number_of_Routes')
    plt.title('Spread_of_number_of_routes_per_bus_')

    plt.show()

#-------------------------------------------------------------------------------
#Positioning Distances
def Positioning_Distances(transitionDistances):

    objects = [x for x in np.arange(1,len(transitionDistances)+1,1)]
    y_pos = np.arange(len(transitionDistances))
    plt.bar(y_pos, transitionDistances, alpha=0.5)
    plt.xticks(y_pos, objects)
    plt.ylabel('Distance')
    plt.xlabel('Routes')
    plt.title('Transition_Distances_for_each_Route_')

    plt.show()

#-------------------------------------------------------------------------------
#Bus Utilization
def Utilization(busObjects):
    PositioningDistance = 0
    graph_Positioning = [0]*48
    graph_Transporting = [0]*48
    for bus in busObjects:
        bus.Add_PM_Depot()
        PositioningDistance = PositioningDistance+bus.PositioningDistance
        for data in bus.Tour:
            for i in range(1,49):
                divWholeLow= round((data[1]/(30))-0.5,0)
                divWholeHigh= round((data[2]/(30))-0.5,0)
```

```
                remLow= round(data[1]%30,1)
                remHigh=round(data[2]%30,1)

                if divWholeLow < i and divWholeHigh > i:
                    if data[0] == 'p':
                        graph_Positioning[i] += 30
                    else:
                        graph_Transporting[i] +=30
                elif divWholeLow == i and divWholeHigh >= i:
                    if data[0] == 'p':
                        graph_Positioning[i] += 30-remLow
                    else:
                        graph_Transporting[i] +=30-remLow

                elif divWholeLow < i and divWholeHigh == i:
                    if data[0] == 'p':
                        graph_Positioning[i] += remHigh
                    else:
                        graph_Transporting[i] +=remHigh

                elif divWholeLow == i and divWholeHigh == i:
                    if data[0] == 'p':
                        graph_Positioning[i] += remHigh-remLow
                    else:
                        graph_Transporting[i] +=remHigh-remLow


    graph_Positioning = [round(x,2) for x in graph_Positioning]
    graph_Transporting = [round(x,2) for x in graph_Transporting]
    graph_Data=[]
    totTime = 30*len(busObjects)
    for i in range(len(graph_Transporting)):
        graph_Data.append([30*i,round((graph_Transporting[i]/totTime)*100,2),\
        round((graph_Positioning[i]/totTime)*100,2)])
    return graph_Data
```

A.5 GRASP Controler

```
import GRASP
import time
import matplotlib.pyplot as plt
import numpy as np


busses= []
Distance=[]
for i in range(100):
    print(i)
    best = 1000
    bestDistance = 10000000
    endTime = time.time() + 10
    while time.time() < endTime:
        a,b,c,d,e=GRASP.Run(40,[350,250,270,292,228,28,240],[350,250,0,0])
        if a < best or (a == best and b < bestDistance):
            best = a
            bestDistance = b

    busses.append(best)
    Distance.append(round(bestDistance,0))

print(np.average(Distance))
print(np.max(Distance))
print(np.min(Distance))
print(np.std(Distance))
x=len(Distance)
y_pos = np.arange(x)
axes = plt.gca()
axes.set_ylim([17850,18250])
plt.bar(y_pos, Distance, alpha=0.5)
plt.ylabel('Distance')
plt.title('Best Distance of each Iteration ')

plt.show()

print(np.average(busses))
print(np.max(busses))
print(np.min(busses))
print(np.std(busses))
```

A.6 Edit Routes

```
import csv
#-------------------------------------------------------------------------------
#Extract Distance Matrix Data
def Add_or_Remove_Routes(startTime,startLoc,endLoc):
    startTimeOG= startTime
    startTime= Normalized_Time(startTime)
    DistanceMatrixNames = []
    DistanceMatrix = []
    matrix = open('newDistanceMatrix.csv','r')
```

```
    reader = csv.reader(matrix)
    for i in reader:
        DistanceMatrix.append(i)
        DistanceMatrixNames.append(i[0])
    matrix.close()

    #Extract and add/remove Routes
    Routes=[]
    route_csv = open('Friday_Index.csv','r')
    reader = csv.reader(route_csv)
    cntRoutes = 0
    allocated = False
    lastStart = 0
    for i in reader:
        nTime = Normalized_Time(i[1])
        if allocated == False and startTime <=nTime and startTime >= lastStart:
            Routes.append([cntRoutes,startTimeOG,startLoc,endLoc])
            cntRoutes +=1
            allocated = True
        Routes.append([cntRoutes,i[1],i[2],i[3],i[4]])
        cntRoutes +=1
        lastStart = Normalized_Time(i[1])
    route_csv.close()

    #-------------------------------------------------------------------------
    #Create new Routes csv
    cnt =0
    outputFile = open('Friday_Index.csv', 'w')
    for i in Routes:
        if len(i) ==4:
            indLoc = 0
            for loc in DistanceMatrixNames:
                if loc == i[2]:
                    indStart = indLoc
                if loc == i[3]:
                    indEnd = indLoc
                indLoc +=1
            d=DistanceMatrix[indEnd][indStart]
            outputFile.write(str(i[0])+','+i[1]+','+str(indStart)+','+str(indEnd)+','+str(d)+'\n')
        else:
            outputFile.write(str(i[0])+','+i[1]+','+i[2]+','+i[3]+','+i[4]+'\n')

        cnt+=1

    outputFile.close()

def Reset():
    route_csv = open('Friday_Index_OG.csv','r')
    reader = csv.reader(route_csv)
    Routes = []
    for i in reader:
       Routes.append(i)
    route_csv.close()

    outputFile = open('Friday_Index.csv', 'w')
    for i in Routes:
       outputFile.write(i[0]+','+i[1]+','+i[2]+','+i[3]+','+i[4]+'\n')

    outputFile.close()

def Normalized_Time(time):
    time = time.split(":")
    normalized_time = int(time[0])*60+int(time[1])
    return normalized_time

#Add_or_Remove_Routes('04:15:00','Alberton North','Braamfontein')
#Reset()
```

Appendix B

Website

B.1 Flask Main

```
from flask import Flask,render_template,request
import json
import Addapted_Best_Fit_Increasing_Alg as G

app = Flask(__name__)
app.config["DEBUG"] = True

@app.route("/", methods=["GET", "POST"])
def index():
    return render_template("index.html")

@app.route("/Simulation/", methods=["GET", "POST"])
def Sim():
    return render_template("Simulation.html")

@app.route("/signUp/", methods=["GET", "POST"])
def signUp():
    return render_template("signUp.html")

@app.route('/signUpUser', methods=['POST'])
def signUpUser():
    user =  request.form['username'];
    password = request.form['password'];
    return json.dumps({'status':'OK','user':user,'pass':password});

@app.route('/Run', methods=['POST'])
def Run():
    speed =  int(request.form['speed'])
    DayCapacities = [request.form['r1'],request.form['r2'],request.form['r3'],request.form['r4'],request.form['r5'],r
    ,request.form['r7']]
    NightCapacities = [request.form['n1'],request.form['n2'],request.form['n3'],request.form['n4']]
    LB = float(request.form['LB'])
    DB = float(request.form['DB'])
    bus,distance,routes,dCapacities,nCap,data = G.Run(speed,DayCapacities,NightCapacities,LB,DB)
    #data = [[0, 0.0, 0.0], [30, 0.0, 0.0], [60, 0.0, 0.0], [90, 0.0, 0.0], [120, 0.0, 0.0], [150, 0.0, 0.0], [180,
    #return json.dumps({'status':'OK','speed':speed});
    return json.dumps({'status':'OK','speed': speed,'Busses':str(bus),'Distance': round(distance,2), 'Routes': routes

    #return json.dumps({'status':'OK','Distance': speed,'Capacities':DayCapacities})
```

B.2 Simulation User Interface

```
<!DOCTYPE html>
<html lang="en">
<head>

<title>BPJ</title>
  <meta charset="utf-8">
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <link rel="stylesheet" href="http://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css">
  <link href="http://fonts.googleapis.com/css?family=Montserrat" rel="stylesheet" type="text/css">
  <link href="http://fonts.googleapis.com/css?family=Lato" rel="stylesheet" type="text/css">
  <script src="https://ajax.googleapis.com/ajax/libs/jquery/1.12.4/jquery.min.js"></script>
  <script src="http://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/js/bootstrap.min.js"></script>
  <!--<link rel="shortcut icon" href="Images/favicon.ico" type="image/x-icon"/ >-->
  <link rel="shortcut icon" href="{{ url_for('static', filename='favicon.ico') }}">
  <link href="https://fonts.googleapis.com/icon?family=Material+Icons" rel="stylesheet">
  <link href="{{ url_for('static', filename='Stylesheet.css') }}" rel="stylesheet">
  <meta name="description" content="We specialize in Bus Scheduling and Delivery Optimization. Unlike most consultant
 dont ask for any upfront payment. Before you pay us a cent it is only fair that you test our abilities.">
  <meta name="keywords" content="Bus Scheduling, scheduling, Delivery Optimization, optimization, Delivery Optimisation
 optimisation, VRPTW, VRPH">
```

```
  <script type="text/javascript" src="https://www.gstatic.com/charts/loader.js"></script>
</head>
<body>
<nav class="navbar navbar-default">
                <div class="container">
                        <div class="navbar-header">
                                <a class="navbar-brand" href="http://mcgladdery.pythonanywhere.com/">Bus Fleet Sch
                            </div>
                            <ul class="nav navbar-nav navbar-right">
                              <!--<li><a href="index.php#BusFleet">Bus Fleet Scheduling</a></li> -->
                              <li><a href="http://mcgladdery.pythonanywhere.com/Simulation">Simulation Demo</a></
                            </ul>
                </div>
</nav>
<div class="container-fluid text-center bg-1">
        <h1> Simulation Configuration</h1>
</div>


    <div class="section">
      <div class="container">
        <div class="row">
          <div class="col-md-12">
                  <br>
                  <p>This is a demo of Fourier-E's  bus fleet scheduling capabilities. After signing up we will devel
                  like the one below that will allow you to test the effect that adding new routes, changing capacit
                  will have on your bus fleets schedule.
                  </p>
            <form id = 'form' class="form-signin" action="/Run" method="post" role="form">

              <div class="form-group">
                <label class="control-label">Bus Speed</label>
                <select class="form-control" name = "speed" id="speed">
                                <option>25</option>
                                <option>30</option>
                                <option>35</option>
                                <option>40</option>
                                <option>45</option>
                                <option>50</option>
                                <option>55</option>
                                <option>60</option>
                                <option>70</option>
                                <option>80</option>
                                <option>90</option>
                                <option>100</option>
                </select>
              </div>
                        <div class="form-group">
                <label class="control-label">Distance Buffer (multiplying factor )</label>
                <select class="form-control" name = "DB" id="DB">
                                <option>1</option>
                                <option>1.05</option>
                                <option>1.1</option>
                                <option>1.15</option>
                                <option>1.2</option>
                                <option>1.25</option>
                                <option>1.3</option>
                                <option>1.35</option>
                                <option>1.4</option>
                                <option>1.45</option>
                                <option>1.5</option>
                </select>
              </div>
                        <div class="form-group">
                <label class="control-label">Loading Buffer (minutes)</label>
                <select class="form-control" name = "LB" id="LB">
                                <option>0</option>
                                <option>5</option>
                                <option>10</option>
                                <option>15</option>
                                <option>20</option>
                                <option>25</option>
                                <option>30</option>
                                <option>35</option>
                                <option>40</option>
                                <option>45</option>
                </select>
              </div>

            </div>
          </div>
        </div>
      </div>
      <div class="section">
        <div class="container">
          <div class="row">
            <div class="col-md-12">
              <h3>Capacities for Rest Depots</h3>
            </div>
          </div>
        </div>
      </div>
      <div class="section">
        <div class="container">
          <div class="row">
```

```
      <div class="col-md-12">
        <table class="table" id = 'Form'>
          <thead>
            <tr>
              <th>Name</th>
              <th>Current Capacity</th>
              <th>Test Capacity</th>
            </tr>
          </thead>
          <tbody>
                        <tr>
              <td>Dobs</td>
              <td>350</td>
              <td><input type="number" class="form-control" name = "r1" value = 350></td>
            </tr>
            <tr>
              <td>Nancefield</td>
              <td>250</td>
              <td><input type="number" class="form-control" name = "r2" value = 250></td>
            </tr>
            <tr>
              <td>KyaSand</td>
              <td>170</td>
              <td><input type="number" class="form-control" name = "r3" value = 170></td>
            </tr>
            <tr>
              <td>Linbro Park</td>
              <td>192</td>
              <td><input type="number" class="form-control" name = "r4" value = 192></td>

            </tr>
                          <tr>
              <td>Strydom Park</td>
              <td>28</td>
              <td><input type="number" class="form-control" name = "r5" value = 28></td>
            </tr>
            <tr>
              <td>Steeldale</td>
              <td>8</td>
              <td><input type="number" class="form-control" name = "r6" value = 8></td>
            </tr>
                          <tr>
              <td>Selby</td>
              <td>40</td>
              <td><input type="number" class="form-control" name = "r7" value = 40></td>
            </tr>
          </tbody>
        </table>
      </div>
    </div>
  </div>
</div>

        <div class="section">
  <div class="container">
    <div class="row">
      <div class="col-md-12">
        <h3>Capacities for Overnight Depots</h3>
      </div>
    </div>
  </div>
</div>
    <div class="section">
  <div class="container">
    <div class="row">
      <div class="col-md-12">
        <table class="table">
          <thead>
            <tr>
              <th>Name</th>
              <th>Current Capacity</th>
              <th>Test Capacity</th>
            </tr>
          </thead>
          <tbody>
                      <tr>
              <td>Dobs</td>
              <td>350</td>
              <td><input type="number" class="form-control" name = "n1" value = 350></td>
            </tr>
            <tr>
              <td>Nancefield</td>
              <td>250</td>
              <td><input type="number" class="form-control" name = "n2" value = 250></td>
            </tr>
                          <tr>
              <td>KyaSand</td>
              <td>170</td>
              <td><input type="number" class="form-control" name = "n3" value = 0></td>
            </tr>
                          <tr>
              <td>Linbro Park</td>
              <td>192</td>
              <td><input type="number" class="form-control" name = "n4" value = 0></td>
            </tr>
```

```
                </tbody>
              </table>
            </div>
          </div>
        </div>
      </div>
            <div class="section">
        <div class="container">
          <div class="row">
            <div class="col-md-12">
                        <br>
                        <button  class="btn btn-primary btn-block"> <h3>Run Simulation</h3></button>
              </form>
                        <br>
            </div>
          </div>
        </div>
      </div>

<div class="section">
  <div class="container" style="display: none" id = "resultsDiv">
          <div class="row">
            <div class="col-md-12" >
                  <h2>Results</h2>
                  <p id = "Results">
                  </p>
                  <br>
                  <br>
            </div>
          </div>
          <div class="row">
                  <div class="col-md-12" >
                        <div id="chart_div"></div>
                  </div>
          </div>
          <div class="row">
            <div class="col-md-12">
                    <h3>Day Depot Utilization</h3>
              <table class="table" >
                <thead>
                  <tr>
                    <th>Name</th>
                    <th>Current Capacity</th>
                    <th>Busses Using It</th>
                  </tr>
                </thead>
                <tbody>
                            <tr>
                  <td>Dobs</td>
                  <td>350</td>
                  <td><div id="DC1"></div></td>
                </tr>
                <tr>
                  <td>Nancefield</td>
                  <td>250</td>
                  <td><div id="DC2"></div></td>
                </tr>
                <tr>
                  <td>KyaSand</td>
                  <td>170</td>
                  <td><div id="DC3"></div></td>
                </tr>
                <tr>
                  <td>Linbro Park</td>
                  <td>192</td>
                  <td><div id="DC4"></div></td>

                </tr>
                                <tr>
                  <td>Strydom Park</td>
                  <td>28</td>
                  <td><div id="DC5"></div></td>
                </tr>
                <tr>
                  <td>Steeldale</td>
                  <td>8</td>
                  <td><div id="DC6"></div></td>
                </tr>
                                <tr>
                  <td>Selby</td>
                  <td>40</td>
                  <td><div id="DC7"></div></td>
                </tr>
              </tbody>
            </table>
          </div>
        </div>
            <div class="row">
            <div class="col-md-12">
                    <h3>Overnight Depot Utilization</h3>
              <table class="table">
                <thead>
                  <tr>
                    <th>Name</th>
```

```
                    <th>Current Capacity</th>
                    <th>Test Capacity</th>
                  </tr>
                </thead>
                <tbody>
                            <tr>
                    <td>Dobs</td>
                    <td>350</td>
                    <td><div id="NC1"></div></td>
                  </tr>
                  <tr>
                    <td>Nancefield</td>
                    <td>250</td>
                    <td><div id="NC2"></div></td>
                  </tr>
                                <tr>
                    <td>KyaSand</td>
                    <td>170</td>
                    <td><div id="NC3"></div></td>
                  </tr>
                                <tr>
                    <td>Linbro Park</td>
                    <td>192</td>
                    <td><div id="NC4"></div></td>
                  </tr>

                </tbody>
              </table>
            </div>
          </div>
  </div>
</div>

    <div class="section">
      <div class="container" style = "display:none">
        <div class="row">
          <div class="col-md-12">
                  <h3>Routes</h3>
                       <p>The below table has all the routes that are currently in the bus schedule and 20 routes th
                       updating the required parameter to Yes . The existing routes can be removed by changing the r
                       No. Once you have changed the routes appropriately you can run the simulation.
                       </p>
            <table class="table table-bordered table-hover table-striped">
              <thead>
                <tr>
                  <th>Id</th>
                                <th>Start Time</th>
                  <th>Start Loc</th>
                  <th>End Loc</th>
                                <th>Required?</th>
                </tr>
              </thead>
              <tbody>
                <tr>
                  <td>1</td>
                  <td>Mark</td>
                  <td>Otto</td>
                  <td>@mdo</td>
                                <td>Yes</td>
                </tr>
                <tr>
                  <td>2</td>
                  <td>Jacob</td>
                  <td>Thornton</td>
                  <td>@fat</td>
                </tr>
                <tr>
                  <td>3</td>
                  <td>Larry</td>
                  <td>the Bird</td>
                  <td>@twitter</td>
                </tr>
              </tbody>
            </table>
          </div>
        </div>
      </div>
    </div>

 <script>
google.charts.load('current', {packages: ['corechart', 'bar']});
google.charts.setOnLoadCallback();

$(function() {

    $('button').click(function() {
                event.preventDefault();
        $.ajax({
            url: '/Run',
            data: $('#form').serialize(),
            type: 'POST',
            success: function(response) {

                                res = JSON.parse(response);
                                //console.log(response);
```

```
                                    document.getElementById('Results').innerHTML = 'We used <b>' + res.Busses +' busses</
                                    ' kilometres </b> to execute all '+res.Routes+' routes.';
                                    $('#resultsDiv').show( "slow" );

                                    dayCapacities = res.Capacities;
                                    document.getElementById('DC1').innerHTML = dayCapacities[0];
                                    document.getElementById('DC2').innerHTML = dayCapacities[1];
                                    document.getElementById('DC3').innerHTML = dayCapacities[2];
                                    document.getElementById('DC4').innerHTML = dayCapacities[3];
                                    document.getElementById('DC5').innerHTML = dayCapacities[4];
                                    document.getElementById('DC6').innerHTML = dayCapacities[5];
                                    document.getElementById('DC7').innerHTML = dayCapacities[6];

                                    nightCapacities = res.nCap;
                                    document.getElementById('NC1').innerHTML = nightCapacities[0];
                                    document.getElementById('NC2').innerHTML = nightCapacities[1];
                                    document.getElementById('NC3').innerHTML = nightCapacities[2];
                                    document.getElementById('NC4').innerHTML = nightCapacities[3];

                                    location.hash = "#Results";
                                    var Gdata = res.Data;
                                    drawStacked(Gdata);

            },

        });
    });
});

function drawStacked(Gdata) {
      var data = new google.visualization.DataTable();
      data.addColumn('number', 'Time of Day');
      data.addColumn('number', 'Transporting');
      data.addColumn('number', 'Positioning');

      data.addRows(Gdata);
          //console.log(Gdata);
      var options = {
        title: 'Bus Utilization Throughout the Day',
        isStacked: true,
        hAxis: {
          title: 'Time of Day (min past 00:00)',
          viewWindow: {
            min: [7, 30, 0],
            max: [17, 30, 0]
          }
        },
        vAxis: {
          title: 'Utilization'
        }
      };

      var chart = new google.visualization.ColumnChart(document.getElementById('chart_div'));
      chart.draw(data, options);
    }
</script>
</body>
</html>
```

Appendix C

Industry Sponsorship Form

Department of Industrial & Systems Engineering
Final Year Projects
Identification and Responsibility of Project Sponsors

All Final Year Projects are published by the University of Pretoria on *UPS a*nd thus freely available on the Internet. These publications portray the quality of education at the University and have the potential of exposing sensitive company information. It is important that both students and company representatives or sponsors are aware of such implications.

Key responsibilities of Project Sponsors:

A project sponsor is the key contact person within the company. This person should thus be able to provide the best guidance to the student on the project. The sponsor is also very likely to gain from the success of the project. The project sponsor has the following important responsibilities:

1. Confirm his/her role as project sponsor, duly authorised by the company. Multiple sponsors can be appointed, but this is not advised. The duly completed form will considered as acceptance of sponsor role.
2. Review and approve the Project Proposal, ensuring that it clearly defines the problem to be investigated by the student and that the project aim, scope, deliverables and approach is acceptable from the company's perspective.
3. Review the Final Project Report (delivered during the second semester), ensuring that informationis accurate and that the solution addresses the problems and/or design requirements of the defined project.
4. Acknowledges the intended publication of the Project Report on UP Space.
5. Ensuresthat any sensitive, confidential information or intellectual property of the company is not disclosed in the Final Project Report.

Project Sponsor Details:

Company:	**Fourier E Consolation Services (Pty) Ltd for Putco**
Project Description:	Develop a bus schedule for Putco that satisfies all constraints and outperforms the current one. (minimize distance and number of busses)
Student Name:	**Duncan McGladdery**
Student number:	**13032969**
Student Signature:	
Sponsor Name:	**Werner Schoeman**
Designation:	**Manager: Operational Design and Measurement; Product Development**
E-mail:	**schoemanw@fourier.co.za**
Tel No:	**082 876 6314**
Cell No:	**082 876 6314**
Fax No:	
Sponsor Signature: