OPTIMAL JOB ASSIGNMENT FOR A HOME-HELP SERVICE

Evdokia Mina

C1544038

School of Computer Science and Informatics

ABSTRACT

The aim of this project is to further research into the areas of Computational Optimisation with Linear Programming to solve a variant of the well-known Traveling Salesman Problem in a specific scenario. This scenario consists of a company that schedules home-help visits for elderly patients, which requires the assignment of employees to elderly people in such a way that the total distance travelled is minimal. Each employee starts their journey from their own house, visit patients in their houses and finish where they started. The staff is made up of a fixed number of support workers that need to visit a fixed number of homes. In addition, the job assignment should be fair, meaning that each employee must have approximately the same amount of workload and no employee must be left out (when the number of employees is less than or equal to the number of patients). This paper explores the ways of developing and implementing an optimal solution. For the sake of this paper, all data used is randomly generated and not related to real people or figures of any kind.

ACKNOWLEDGEMENTS

This project would have not been possible without the constant help of Dr Richard Booth, who provided me with the required knowledge and tools to evolve and succeed in my academic studies. Moreover, he was always willing to resolve any issues or queries I had which is greatly appreciated.

TABLE OF CONTENTS

TABLE OF FIGURES

LIST OF TABLES

1 INTRODUCTION

1.1 COMPUTATIONAL OPTIMISATION AND LINEAR PROGRAMMING

There are many computational optimisation problems that are explored daily by many scientists. In this paper, we explore the use of computational optimisation and linear programming to construct a model that solves a real-life problem. Computational optimisation is a mathematical technique that consists of maximising or minimising (or even some more complex operations) of certain functions for better decision making, which was inherited from applied mathematics [1]. Linear Programming was later introduced to solve special cases of computational optimisation problems that are expressed as linear equations in order to produce the best results [2]. Linear programming consists of constraints in the form of linear equations (e.g. $x + y \geq 4$) that deal with equalities or inequalities in order to limit non-optimal solutions. In practice, these functions are linear vectors on two-dimensional surfaces, also known as planes, where constraints are areas on the surface. Vectors are later solved to find their intersecting coordinates within the given area.

1.2 PROJECT OVERVIEW

The lack of research in the specific variation discussed in this paper made this project a lot more interesting. Not only was there not enough documentation on similar variations, but even less documentation on an implemented complex computational optimisation model similar to the one we explored. All employees must be assigned to all patients and a patient can only be visited once and never again. Each employee must start and finish in their own house. The cost of travel must be minimised and only one employee can be involved in each subtour generated. Imagine having a set of employees E with the red circles denoting their location and a set of clients N with black circles denoting their locations. An example of correct assignment would be as follows:

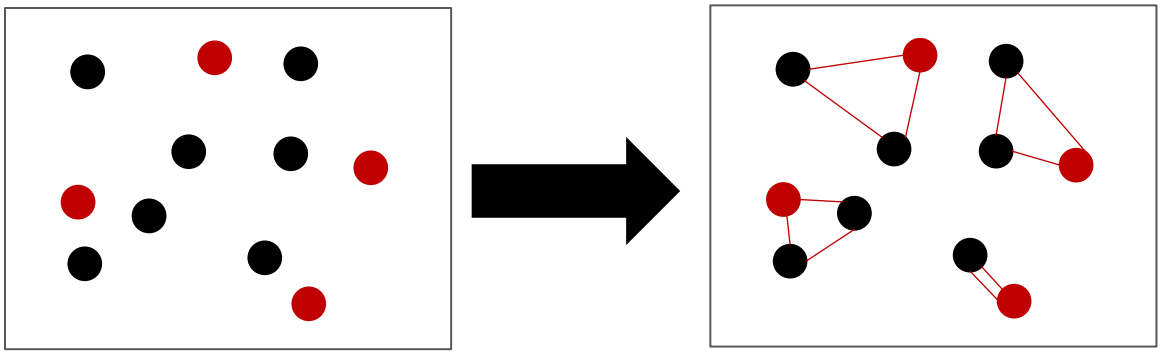

Figure 2-Pre-assigned Employees and Clients

Figure 1- Correct Job Assignment

An example of an incorrect job assignment would be:

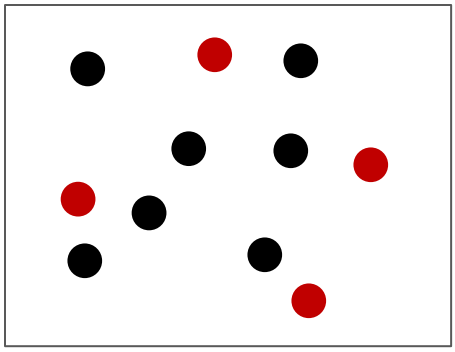

Figure 3-Pre-assigned Employees and Clients

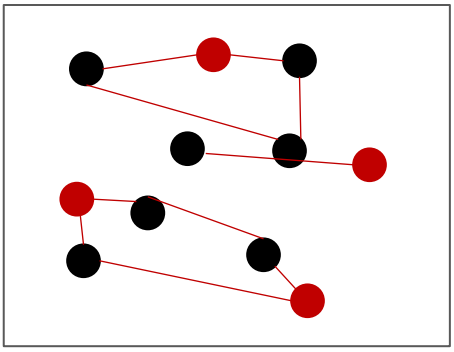

Figure 4-Incorrect Job Assignment

This project sought to develop and implement an algorithm to solve this real-world problem using Integer Linear Programming. The algorithm should be able to find an optimal solution (or a feasible solution) to this problem, taking into consideration the fair workload assignment.

2 BACKGROUND RESEARCH

2.1 TRAVELLING SALESMAN PROBLEM

2.1.1 Introduction

The classic Travelling Salesman Problem (TSP) is one of the most studied and discussed computation optimisation problems in the world. This problem was first mathematically introduced by the Irish mathematician W.R Hamilton and the British mathematician Thomas Kirkman [3]. Today, a lot of scientists use the Travelling Salesman Problem to formulate variations of it and come up with different techniques on how to solve it. It is mainly used in transportation and logistics applications and many other complex optimisation problems with additional constraints. It consists of one salesman travelling from one specified city (or depot) to a set of selected cities C in such a way that the total cost c_{ij} travelled is minimal.

2.1.2 Problem Explanation

The approach taken to solve this problem is also known as the Dantzig-Fulkerson-Johnson formulation [3] but can be solved in many different ways.

$$\boldsymbol{Minimise} \sum_{i,j \in A} c_{ij} x_{ij}$$

s.t.

$$\sum_{j \in V} x_{ij} = 2, \qquad \forall i \in V \tag{1}$$

$$\sum_{i,j \in S, i \neq j} x_{ij} \leq |S| - 1\,, \qquad \forall\, S \subset C, S \neq \emptyset \tag{2}$$

$$x = \begin{cases} 1, & if\ edge\ (i,j) \in A\ is\ selected \\ 0, & if\ edge\ (i,j) \in A\ is\ not\ selected \end{cases} \tag{3}$$

The 2-degree constraint (1) [4] shows that each city must only be visited once and never again, where V is the set of all cities. Subtour elimination constraint (2) [4] ensures that no subtours S are included in the solution. The variable x_{ij} (3) is a binary variable that takes the value 0 if an edge is not selected and if 1 an edge is selected where the set A consists of all possible edges i, j between all the cities where i is the current city and j is the next city to be visited.

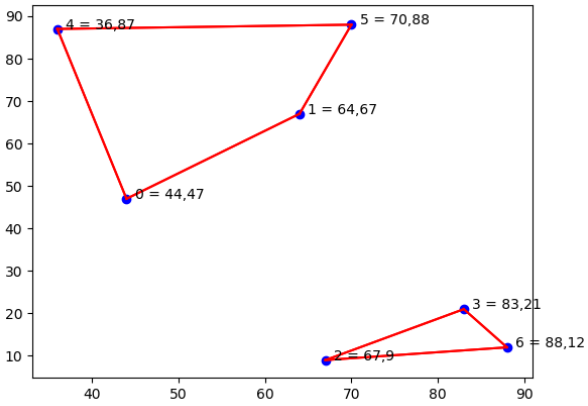

Figure 5-Incorrect TSP, without subtour elimination

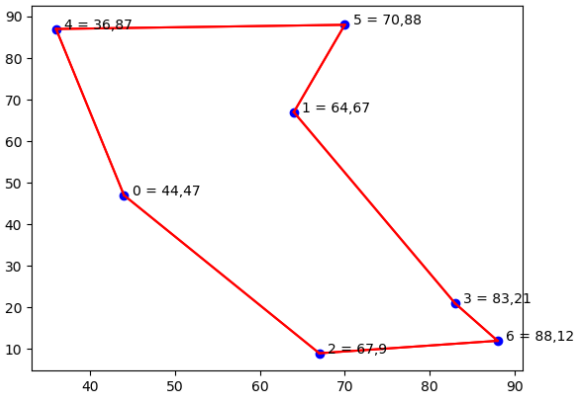

Figure 6-Correct TSP, with subtour elimination

2.1.3 Application

In application to this project, it is only one specified scenario where only one employee is used, and all the patients must be visited. For the purpose of this project, the Travelling Salesman Problem can be of help to the problem discussed in this paper by using the subtour elimination constraint (2) but it is not an ideal solution.

2.2 MULTI TRAVELLING SALESMAN PROBLEM

2.2.1 Introduction

There have been further studies on how to improve the classical Travelling Salesman Problem and the first variation is having multiple travelling salesmen (mTSP). Given a set of cities C, there are m salesmen that need to travel from one specified depot to all the cities. The objective remains the same; the total cost travelled by all salesmen must be minimised. In addition, each city must only be visited once by a salesman.

2.2.2 Explanation

Just like the classic TSP, the mTSP uses a set of cities C, a set of travelling salesmen m and a set of all possible edges between the cities A. To solve this, we use following formulation [5]:

$$\boldsymbol{Minimise} \sum_{i,j \in A} c_{ij} x_{ij}$$

s.t.

$$\sum_{j \in V} x_{1j} = m \tag{1}$$

$$\sum_{j \in V} x_{j1} = m \tag{2}$$

$$\sum_{j \in V} x_{ij} = 1, \qquad \forall i \in V \tag{3}$$

$$\sum_{i \in V} x_{ij} = 1, \qquad \forall j \in V \tag{4}$$

$$u_i - u_j + p \cdot x_{ij} \leq p - 1, \qquad \forall 2 \leq i \neq j \leq n \tag{5}$$

$$x = \begin{cases} 1, & if\ edge\ (i,j) \in A\ is\ selected \\ 0, & if\ edge\ (i,j) \in A\ is\ not\ selected \end{cases} \tag{6}$$

The cost of each edge travelled is denoted by c_{ij} where i is the current city and j is the next city to be visited with $i, j = 1$ denoting the depot. The constraints (1) and (2) ensure that exactly m salesmen leave from and return to the depot. The degree constraints (3) and (4) are added to all the cities to ensure exactly one salesman enter and leaves from the city. The constraint (5) also known as the Miller-Tucker-Zemlin [5] constraint, is used to ensure only a maximum number of cities p is visited by each salesman as subtours are now allowed. The variable u_i indicates the position of a node i in a subtour and u_j denotes the position of the next node in the subtour. Finally, a binary variable x is still used to indicate whether an edge is selected or not.

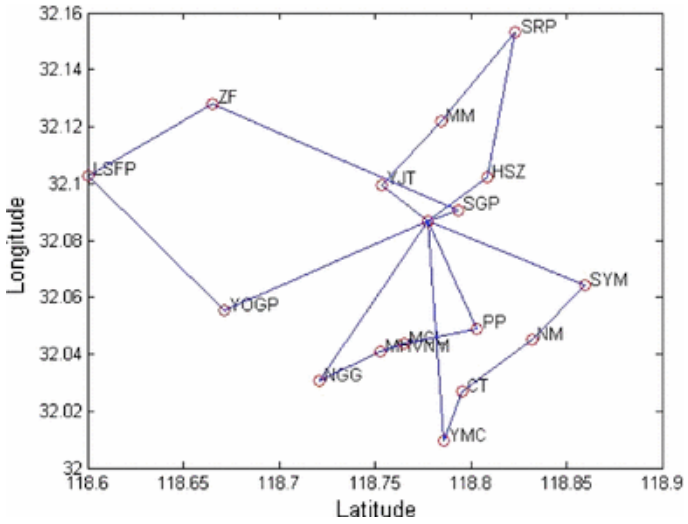

Figure 7 [6]- *Example of a mTSP outcome*

2.2.3 Application

The discovery of the MTZ constraint is one of the most important parts of the background research taken. This constraint explains how fairness takes place in such a way that all salesmen travelling visit roughly the same number of cities while minimising the objective. In the problem discussed in the paper, the salesmen represent the employees and the cities represent the patients in their houses. However, it still includes one downfall; the single depot, which makes this solution not ideal.

2.3 Multi-Depot Multi Travelling Salesman Problem

2.3.1 Introduction

Multi-Depot Multiple Travelling Salesman Problem (MDMTSP) is a variation of the well-known

Travelling Salesman Problem (TSP) and Multiple Travelling Salesman Problem (mTSP). It should be pointed out, that there are many variations of the MDMTSP and each scientist that explores this expresses it in a range of ways. For example, some scientists included vehicles capacities and others don't. Moreover, some add extra variables to indicate depot usage. The paper used to explore this MDMTSP was not only the closest variation to this project but also well and simply written which made it easily understandable. The MDMTSP consists of multiple salesmen and multiple depots. A salesman can start its journey at any of the available depots, visit a set of cities and finish back at the same depot. Each city can only be visited once by a salesman and only one salesman can leave from and return to a depot.

2.3.2 Explanation

The MDMTSP consists of an additional set of depots I along with the previously declared variables for the mTSP in a slight different denotation; set of cities J, set of all possible arcs E, cost denoted by c_{ij} and variable x_{ij}. However, the variable x not only takes the value 0 if the edge (i, j) is not selected and 1 if the edge is selected but also takes the variable 2 in the case where a salesman takes a trip from one depot to a city and back to the depot (also known as a return trip) [7]. The following formulation [7] is a potential solution:

$$\boldsymbol{Minimise} \sum_{i,j \in A} c_{ij} x_{ij}$$

s.t.

$$x\big(\delta(j)\big) = 2, \qquad \forall j \in J \tag{1}$$

$$x\big(\gamma(S)\big) \leq |S| - 1, \qquad \forall S \subseteq J \tag{2}$$

$$\sum_{i \in I'} x_{ij} + 2x\big(\gamma(S \cup \{j,l\})\big) + \sum_{k \in I \setminus I'} x_{kl} \leq 2\,|S| + 3, \qquad \forall\, j,l \in J$$

$$S \subseteq J\{j,l\}, S, \neq \emptyset; I' \subset I \tag{3}$$

$$\sum_{i \in I'} x_{ij} + 3x_{ij} + \sum_{k \in I \setminus I'} x_{kl} \leq 2\,|S| + 3, \qquad \forall\, j,l \in J\,, I' \subset I \tag{4}$$

$$x_{ij} \in \{0,1,2\}\ \forall i \in I, \forall j \in J \tag{5}$$

$$x_{ij} \in \{0,1\}\ \forall i \in J, \forall j \in J \tag{6}$$

The constraint (1) is the normal 2-degree constraint that ensures every city is only visited ones and never again by a salesman. Next, the constraint (2) ensures that no subtours within a subtour solution are further generated. One of the most important constraints used (3) (Figure 8) eliminates more than one salesman travelling the same subtour [7]. Thus, a solution including $i_1, j_1, \dots, j_t, i_2$ where i being the depot with $I' = i_1$ the first depot where the salesman has travelled from, $j = j_1$ the first visited city, $l = j_t$ the last visited city and $S = \{j_2, \dots, j_{t-1}\}$ the set of cities visited excluding the first and last cities [7]. The constraint (4) deals with return tours, which occur when a salesman visits a city and immediately returns to the same depot [7]. Finally, constraints (5) and (6) are responsible for the values x_{ij} take depending on the case (if it is a return tour) and

whether it is an active edge or not.

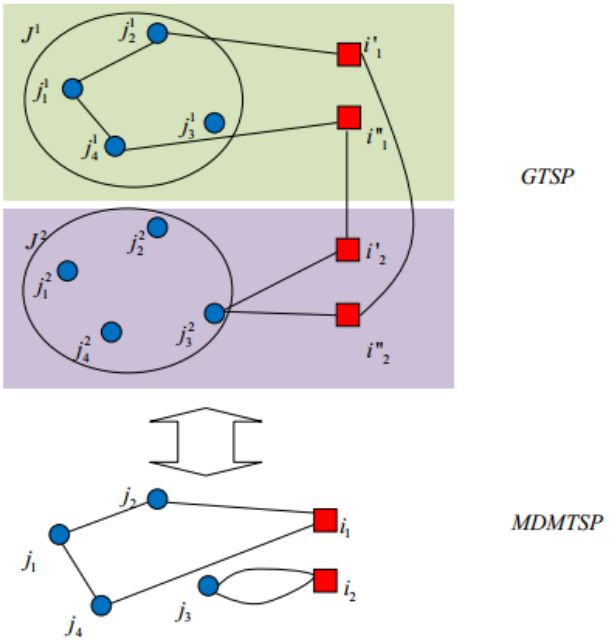

Figure 8 [7]- *MDMTSP constraint illustrated*

2.3.3 Application

This is the closest approach to the project explored in this paper but is still subject to issues and imperfections that lead to wrong solutions. The reason why this approach generates wrong solutions is that it does not take into consideration that all depots must be used. Although, all cities are considered by adding the 2-degree constraint (1) on the cities a lot of the time multiple depots are left unused. In this project, all employees must always be used. However, adding a 2-degree constraint on the travelling salesmen was not an option as it would generate an infeasible model. An infeasible model can no longer be solved, and no solution is generated. Moreover, this approach raises some logical programming issues. The constraints used (5) and (6), deal with variables being binary and continues at the same time which programmatically is incorrect and needed to find ways to overcome human subconscious assumptions. Finally, this solution did not take into consideration having more than two travelling salesmen in a subtour which is again a problem for the model we are trying to generate.

3 APPROACH

3.1 PROJECT STRUCTURE

The steps to be taken for the development of the project needed to be established. Now that the problem has been identified and well understood, it was time to invent a plan on how to approach this and solve it. All progress was logged using a Gantt chart to ensure constant development and improvement.

The world of computational optimisation is constantly evolving and available tools on the market that help with this process are always improving. The first stage of this project was to research and identify the best tool that will be used to develop a decent solution to this problem. This tool needed to be easily understood, so that development does not take a lot of unnecessary time. Moreover, it needed to be well documented to be able to deal with complex constraints. Picking a well-fitted tool would not only help the development process but also any future updates, making any project alterations a lot more convenient.

Once the main tools for the development part of the project were identified, the next stage of the project consisted of creating an algorithm with random numbers representing house positions. This phase was extremely important as it is the base of this project and without it, we could not move to the third stage which was making this algorithm more realistic by using dummy data that includes real-life coordinates and randomly generalised names. This stage is continuously repeated until the solution generated by the optimisation model is to a satisfactory level with all the constraints taken into consideration.

After, it was time to bring the algorithm to more realistic standards using dummy data. As this is a project regarding no real-life company or real-life people, all data generated is random and links to no real situation. To show how this could be a real product that companies can use, a database on the Computer Science School's server was created to host the dummy data.

Next, it was time to create a small interface to present the results from the algorithm which included a list of the assignments between employees and patients. To do so, the tools to be used to create this needed to be explored and established.

When the final product is complete, the final stage consists of thoroughly testing it and evaluating it. Each stage was repeated until the satisfactory level was achieved.

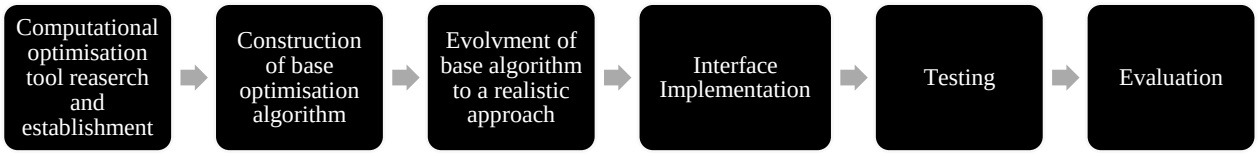

Figure 9- Project Structure

3.2 MAIN VARIABLES AND ASSUMPTIONS

3.2.1 Overview and Formulation

As previously discussed, the aim of this project is to implement a way to assign multiple employees (salesmen) to multiple patients (cities) in such a way that the total amount of distance travelled is minimal. The set of employees is denoted by E and the set of patients by N. For this approach we assume that the number of employees $e = |E|$ and the number of patients $n = |N|$ is always $n \geq e$. Let $G = (V, E)$ be a directed graph where V is a set of vertices such that $V = N \cup E$ and $A = \{(i,j): \forall\, i \in V, \forall\, j \in V\}$ denoting all the arcs between all the vertices in V. The cost of an arc is defined by c_{ij} where i is the beginning node and j is the destination node. Furthermore, we define $Q = n/e$ to be the maximum number of patients to be visited per employee. Each set of subtours generated must include exactly one employee and a maximum of Q patients. All employees and all patients must be assigned. Patients must be visited only once and never again. Each employee must return to their homes (depot) so no employee can start at house A and finish at house B.

We define a variable $x_{ij}, \forall\, (i,j) \in A$ to be binary that takes the value 0 if an arc is not selected to be travelled and 1 is an arc is selected to be travelled. Moreover, we define a variable $u_i\ , \forall i \in V$ to be an integer variable where i denotes the node and has an upper bound (maximum value) of Q. This means that the variable u can take values between $\{0,1, \dots, Q\}$. For a subset $S, S \subseteq V$ we denote $\gamma(S) = \{(i,j) \in A: i,j\ S\}$ [7] where $\gamma(S)$ is the set of all possible arcs within S.

We also define $tours$ to be a set of all generated possible subtour solutions and t is one of the subtours in $tours$ where $t \subseteq V$ and $t \subseteq tours$. It is to be noted that $t = V$ if and only if $e = 1$ meaning that only one employee is available to be assigned to the patients. In addition, we define I to be the set of employees used in a subtour t ,$I \subseteq t, I \subseteq E, I \neq \emptyset$. We denote $SUnion$ to be the set of all patients in a subtour t, $SUnion \subseteq t, SUnion \subseteq N, SUnion \neq \emptyset$. For a subtour $t = \{i_1, j_1, \dots, j_q, i_p\}$ where $i_1, i_2, \dots i_p \in E$ and $j_1, j_2, \dots j_q \in N$, we denote $j = j_1$ the first patient visited. Finally, we denote $I' = i_1$ the first employee in a subtour [7] t.

After a lot of research, we introduce the following formulation:

$$\textbf{\textit{Minimise}} \sum_{(i,j) \in A} c_{ij} x_{ij}$$

s.t.

$$\sum_{j \in V, j \neq i} x_{ij} = 1\ , \qquad \forall i \in N \tag{1}$$

$$\sum_{i \in V, j \neq i} x_{ij} = 1\ , \qquad \forall j \in V \tag{2}$$

$$\sum_{j \in V, j \neq i} x_{ij} = 1\ , \qquad \forall i \in E \tag{3}$$

$$\sum_{i \in V, j \neq i} x_{ij} = 1\,, \qquad \forall j \in E \tag{4}$$

$$if\ x_{ij} = 1 \implies x_{ij} = 0, \qquad \forall\, i,j\ \in A, i\ \in E, j \in E, i\ \neq j \tag{5}$$

$$if\ x_{ij} = 1 \implies u_i - u_j + Q x_{ij} = Q - 1, \qquad \forall\, i,j\ \in A, i\ \notin E, j \notin E, i\ \neq j \tag{6}$$

$$x\big(\gamma(t)\big) \leq |t| - 1, \qquad \forall t \subseteq N, t \neq \emptyset, |I| = 0, |t| > 1 \tag{7}$$

$$\sum_{i \in I'} x_{ij} + \sum_{k \in E \setminus I'} x_{kr} \leq 1, \qquad \forall j \in N, r \in SUnion\,, |I| \geq 2\,, |SUnion| > 0 \tag{8}$$

$$x_{ij} = \begin{cases} 1, & if\ edge\ (i,j) \in A\ is\ selected \\ 0, & if\ edge\ (i,j) \in A\ is\ not\ selected \end{cases} \tag{9}$$

$$u_i = \{0,1,2,\dots,Q\}, \qquad \forall i \in V \tag{10}$$

3.2.2 Formulation Explanation

The aim of the project is to minimise the total cost travelled therefore the project's objective remains the same as TSP (section 2.1), mTSP (section 2.2) and MDMTSP (section 2.3) mentioned prior.

The first four constraints (1), (2), (3) and (4) are the known degree constraints. Because this is a directed problem, as we do care about which client is visited first, constraint (1) ensure that there is only one arc going from a patient to another node in V. Constraint (2) ensures that there is only one arc connecting two nodes. Similarly, constraint (3) ensures that only one employee enters another node in V where constraint (4) ensures that only one employee leaves from a node in V. Next, constraint (5) eliminates the possibility of generating tours with an edge going from one employee directly to another employee. Constraint (6) is the first meaningful constraint which secures a fair workload. This constraint works by keeping a count of the number of patients in each tour. Let's assume that the maximum number of patients to be visited is $Q = 2$, starting at patient 0 $u_0 = 0$ then patient 1 will be $u_1 = 1$, patient 2 in line will be $u_2 = 2$ and as this is the maximum capacity there cannot be a fourth patient in the tour as $u_3 > Q$. It is essential to state that this constraint is not subject to ensuring that the subtour generated is 'correct' just responsible for the number of patients included.

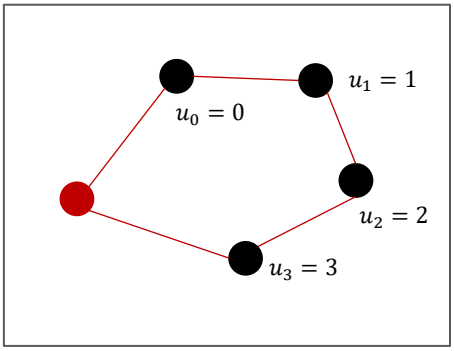

Figure 10- Subtour without fair workload constraint (6)

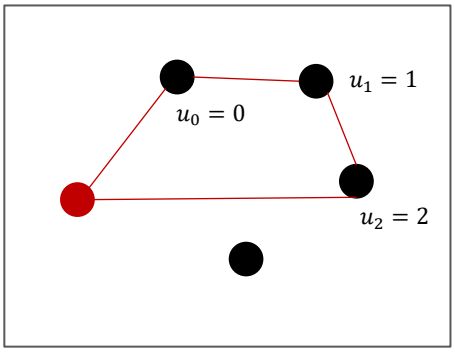

Figure 11- Subtour with fair workload constraint (6)

The following two constraints (7) and (8) are the lazy subtour eliminating constraints which are added every time a solution is violated. To begin, constraint (7) is the well-known subtour elimination constraint but this time works slightly differently to the classic TSP. A subtour solution t is checked to ensure that no subtours exist without any employees.

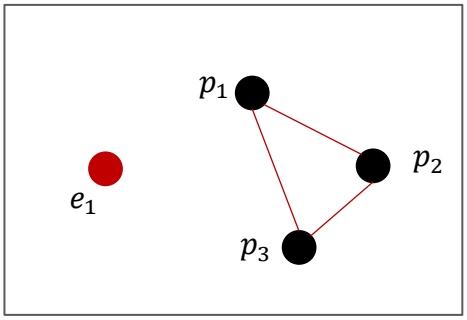

Figure 13-Subtour without constraint (7), Red node denoting employee, Black nodes denoting patients

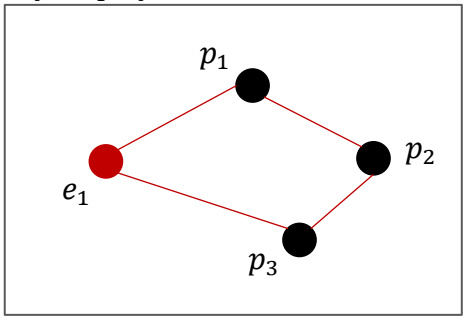

Figure 12- Subtour with constraint (7), Red node denoting employee, Black nodes denoting patients

To continue, we added constraint (8) which is responsible for several inequalities. Firstly, it eliminates solutions where an employee visits exactly one patient and pass through more than one employee's houses. Let's assume subtour $t = \{e_1, p_1, e_2\}$ where $E = \{e_1, e_2, e_3\}$, then $x_{e_1p_1} + x_{e_2p_1} + x_{e_3p_1} = 2 > 1$, therefore such solutions cannot hold.

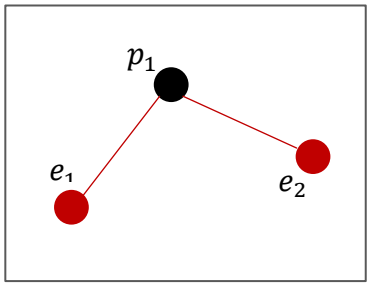

Figure 14-Subtour without constraint (8), Red nodes denoting employees, Black node denoting patient

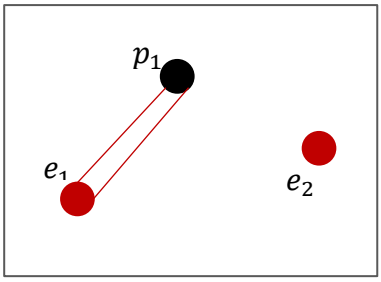

Figure 15-Subtour with constraint (8), Red nodes denoting employees, Black node denoting patient

In a very similar way, this constraint is applied for cases with two or more employees and more than two patients in a subtour t. A solution including a subtour $t = \{e_1, p_1, p_2, p_3, e_2\}$ where $E = \{e_1, e_2, e_3\}$ then $x_{e_1p_1} + x_{e_2p_1} + x_{e_2p_2} + x_{e_2p_3} = 2 > 1$, therefore such solutions are forbidden.

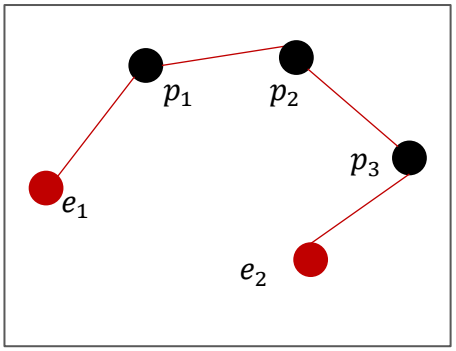

Figure 17-Subtour without constraint (8), Red nodes denoting employees, Black nodes denoting patients

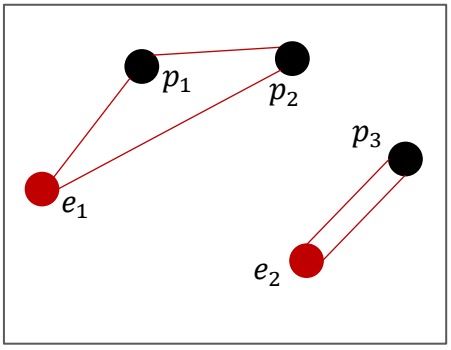

Figure 16-Subtour with constraint (8), Red nodes denoting employees, Black nodes denoting patients

Furthermore, constraint (8) is used to eliminate all possible solutions that include more than two employees within a subtour. If a subtour generated includes a solution such that $t = \{e_1, p_1, e_2, p_2, e_3\}$ and $E = \{e_1, e_2, e_3\}$ then $x_{e_1p_1} + x_{e_2p_1} + x_{e_2p_2} + x_{e_3p_1} + x_{e_3p_2} = 4 > 1$ which cannot be selected.

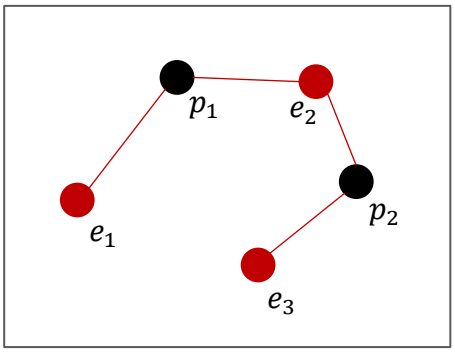

Figure 18-Subtour without constraint (8), Red nodes denoting employees, Black nodes denoting patients

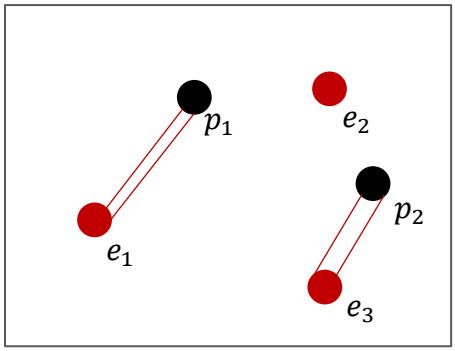

Figure 19-Subtour with constraint (8), Red nodes denoting employees, Black nodes denoting patients

Finally, constraints (9) and (10) define the two variables used throughout the computational model. Constraint (9) is the binary variable x that takes the value 1 if an arc is selected or 0 if not. Then, constraint (10) defines the variable u that takes the values from 0 up to Q, which is the maximum number of clients to be visited.

It is important to note that the above formulation does not handle cases where the number of employees is bigger than the number of clients. Additionally, the figures used to explain the constraints do not show a complete job assignment but only how a subtour is handled; no employees or clients are left alone in the final solution.

4 IMPLEMENTATION

4.1 TOOLS USED

To implement a computational optimisation model using integer linear programming we used Gurobi [8] within Python. Gurobi is a computational optimisation tool used by many companies across the globe to help find solutions automatically with zero to no user intervention. It states to be one of the best products currently in the market as it is free for academic use, provides a lot of help and documentation for Mixed Linear Problems as well as multiple language support including Python, C, C++, Java, MATLAB, .NET and R. Moreover, this tool uses the branch and bound technique to solve the linear models created by the users, such as the one we are studying in this paper. To explain this technique further, when solving a Linear Problem, if the solution produced is an optimal solution, which means that the solution is correct and within the constraint boundaries set, then the branch a bound algorithm stops. Otherwise, the solution found is analysed and extra constraints are added to exclude such a solution in the future. Suppose a value x that must be an integer but the optimisation model returns a value that is float such as 2.3, then we add some extra boundaries such as $x \leq 2$ and $x \geq 3$ to ignore these outcomes [9]. This variable x is now a branched variable and two sub-Linear Programs have been produced where when solved the better of the two solutions is selected as it will continue to be optimal to the main optimisation model [9]. If no solution is feasible, then this heuristic technique of brunch and bound continues until an optimal solution is found.

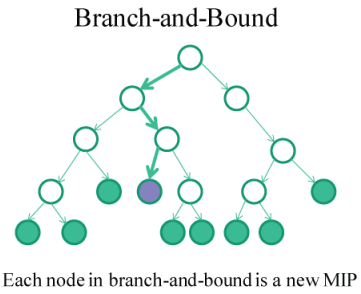

Figure 20 [9]- *Branch and Bound example*

To use this tool, we registered with Gurobi to allow the download of the Gurobi Software. We ensured that we use this for academic reasons only. Then we installed the Gurobi library in python using the "pip" command. To write and develop the code we used Visual Studio Code alongside GitHub to keep track of changes in the code and have a backup in case of system failure.

Moreover, MySQL Workbench was used to manage the data stored by a potential company for the algorithm to use. It is a tool that helps visualise relational data and provides many features such as data modelling, server configuration, backups and others and help create, execute and optimise complex queries [10]. In addition, to show the results the kivy [11] package was used within Python. Kivy is an open source library that allows the development of interesting and contemporary interfaces. It works well across many platforms as well as giving a lot of freedom to the developer to create something interesting.

4.2 PROGRAM STRUCTURE

The following figures show how this project was structured and were generated using an online platform [12].

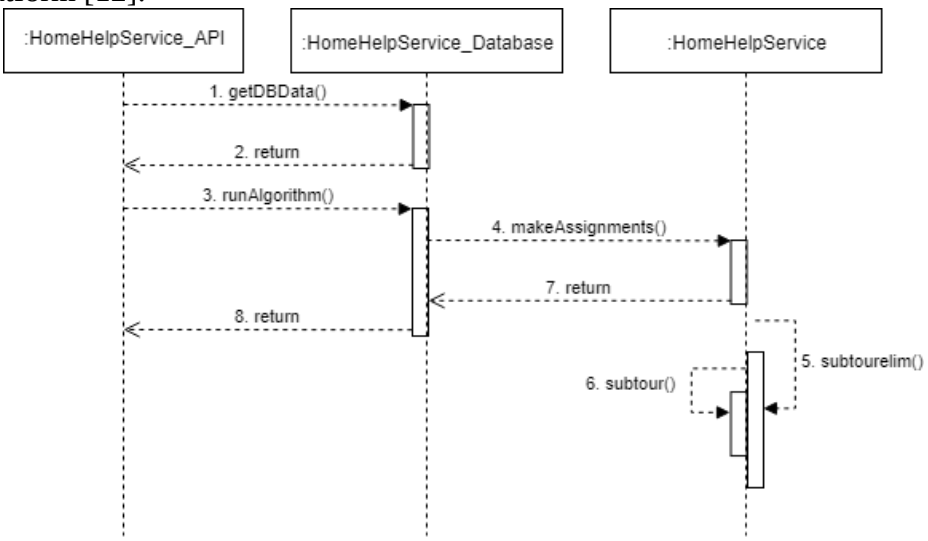

Figure 21-Sequential Flow Diagram showing the calls between the three files

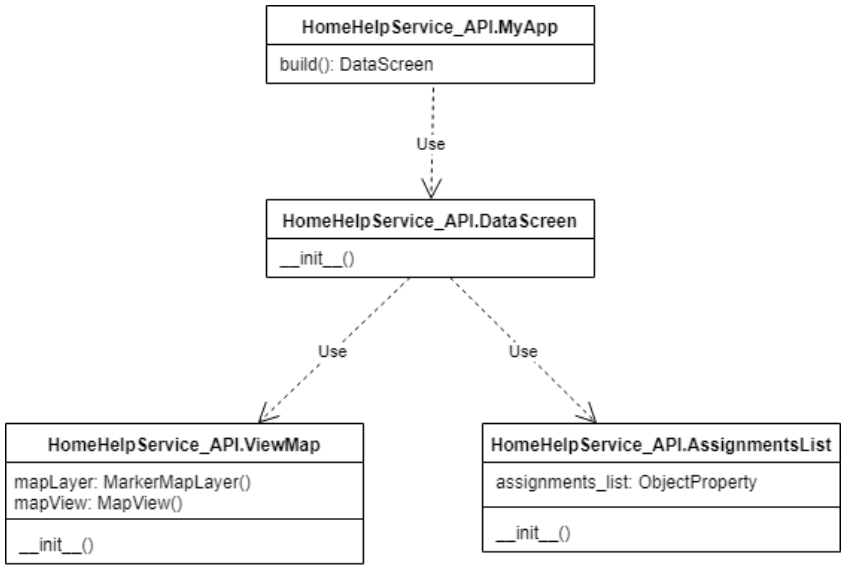

Figure 22-UML diagram of the HomeHelpService_API file

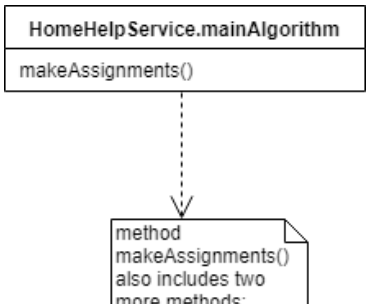

Figure 23-UML diagram of the HomeHelpService file

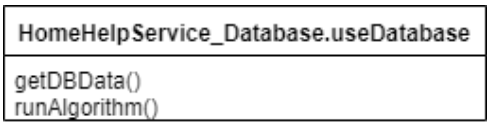

Figure 24- UML diagram of the HomeHelpService_Database file

In Figure 21 we can see that there are three main files; HomeHelpService_API, HomeHelpService_Database and HomeHelpService. The HomeHelpService_API holds everything that deals with the front end of the software. The HomeHelpService_Database is responsible for attempting the database connection. Finally, HomeHelpService holds the main algorithm which generates the subtours of all employees by making the job assignments.

Lastly, Figure 22, Figure 23 and Figure 24 show how each component is set up and how the classes are constructed including the methods they use with the help of UML diagrams.

4.2.1 Job Assignment Code Overview and Explanation

To begin we initialise the number of employees and number of clients used. As the code is run on a not so powerful computer, we set a limit of maximum employees and maximum clients to be 10. It is essential to note that each employee and each client have a different number to establish them. For example, $N = \{0,1,2,3,4,5,6,7,8\}$ and $E = \{9,10,11,12\}$ therefore $V = \{0,1,2,3,4,5,6,7,8,9,10,11,12\}$ is a valid set of employees and clients but $N = \{0,1,2,3,4,5,6,7,8\}$ and $E = \{0,1,2,3,4\}$ is not valid as $V = \{0,1,2,3,4,5,6,7,8\}$ which eliminates the employees. Then we generate random locations for each employee and client and store them in a dictionary where the key denotes the node and the value is a tuple of x and y coordinates e.g. $locations = \{\{0: (51.84001, 167.0707),\ 1: (18.8132523, 43.25057),\ 2: (37.8673962, 131.691011)\}$. The cost is then calculated by computing the distance between each pair of points stored in a dictionary such that $c = \{(0,1): 10, (0,2): 4, (1,2): 3, (1,0): 10, (2,0): 4, (2,1): 3\}$. The Google's Distance Matrix API [13] is used to automatically calculate the distance between two given points by initiating a connection using a certified key provided by Google when creating an account and a project on their cloud platform. However, the Google APIs have a usage limit as each connection request costs money and the monthly free credit allowance provided by Google is limited. If no credit is available in the Google account, then the distance is calculated using the known hypotenuse function in python "$numpy.hypot()$". The motivation behind using the Google API is to gain a more accurate calculation of the distance as often enough distance between two houses is not a straight line.

Next, we create a computational optimisation model named "m" using the "Model()" function that Gurobi provides. To continue, we established the two variables used; x as a binary variable and u as an integer variable with an upper bound of Q. We then proceed by setting the objective which minimises the total cost travelled as mentioned in section 3.2 of the paper. Later we set the first six main constraints ((1), (2), (3), (4), (5) and (6) from section 3.2.1). We added these as part of the main body of the code as we always want these constraints to be considered because a valid solution can be generated using just these specifications without any lazy constraints. Once these are added, we set the "LazyConstraint" parameter to be equal to 1 so that lazy constraints can be added later when the model is optimised. A lazy constraint is an "extra" constraint that is used to eliminate solutions when the output is not what was expected but they are not required as a valid solution can be generated without them. Furthermore, cutting planes must be controlled so that no feasible solution is dismissed. Cutting planes is a method used in Mixed Integer Linear Programming in order to allow finding a new optimal point which is then tested for being an integer solution that satisfies all inequalities [14]. A cut can be added to relaxed models as well [14]. However, we do not want this as it could eliminate valid solutions and substitute them with an invalid one. In this computational model, the "Cuts" parameter is set to 0 to shut off all cuts. By

default, Gurobi sets the "Cuts" parameter to -1, which means that the optimisation model will automatically choose whether it will shut off all cuts, perform a moderate cut, generate an aggressive cut or a very aggressive cut [15].

Eventually, it is time to call the "optimise(subtourelim)" function where "optimise()" is a build in method within Gurobi that helps solve the model and "subtourelim()" is a custom subtour elimination callback method. The "subtourelim()" function checks if a Mixed Integer Linear solution was generated and adds the lazy constraints (7) and (8) if necessary. It starts by getting the edges that are active (have value 1) and then calls a second custom function "subtour(edges)" which takes the set of active edges as a parameter and performs a set of operations to get the generated subtours in a form of a list. Assume a set of nodes $V = \{0,1,2,3,4,5,6,7,8\}$ where $E = \{6,7,8\}$ and $N = \{0,1,2,3,4,5\}$, if a set of active edges is $[(0,3),(1,8),(2,1),(3,6),(4,0),(5,7),(6,4),(7,5),(8,2)]$ then the generated subtour is $[[8,2,1],[7,5],[6,4,0,3]]$ which is returned by the "subtour(edges)" function. Henceforth, the patients and employees are separated and stored into two lists $Sunion$ and I respectively. The tours generated are ultimately checked for two cases; there are no employees (7) in the subtour or have more than one employee (8). It is key to realise that only one of the lazy constraints is added at a time. Gurobi cannot handle multiple inequalities and therefore only one lazy constraint must be added for each violated integer solution as solutions cannot have 0 employees and more than 1 employees simultaneously. To do so, an if loop was used. Firstly, the generated subtour solution is checked for lack of employees. If it does, in fact, have no employees then constraint (7) as mentioned in section 3.2.1 is added. On the other hand, if the subtour generated includes more than one employee then constraint (8) mention in section (3.2.1) is added. Otherwise, neither are added to the model as a correct subtour solution was generated.

While implementing, we tried splitting the lazy constraint (8) into three separate scenarios as follows:

$$\sum_{i \in I'} x_{ij} + \sum_{k \in E \setminus I'} x_{kj} \leq 1, \qquad \forall j \in N, |I| \geq 2, |SUnion| = 1 \tag{11}$$

$$\sum_{i \in I'} x_{ij} + \sum_{k \in E \setminus I'} x_{kr} \leq 1, \qquad \forall j \in N, r \in SUnion\,, |I| \geq 2\,, |S| > 0 \tag{12}$$

$$\sum_{i \in I'} x_{ij} + 3x_{jl} + \sum_{k \in E \setminus I'} x_{kl} \leq 4, \qquad \forall j, l \in N, |I| \geq 2, |SUnion| = 2 \tag{13}$$

The solution was checked again in the same manner as before. The first split up constraint (11), deals with scenarios where there is more than one employee but only one patient in the subtour. The second one (12) deals with cases where there are more than 1 employee and more than 2 clients in the subtour and lastly the third constraint (13) deals with scenarios where more than 1 employee is used in a subtour and the number of clients is exactly two. The reason behind this was to separate each violation even further using an if loop. In theory, the optimisation model would work better as only one lazy constraint would be added for each specific violation but as will be discussed below (Section 5), this only makes the program much slower for complex cases.

After, the status of the Mixed Integer Linear Programming model is checked. The current model

status is taken using the "status" method within Gurobi. This method returns a numeric code which has different meanings [16]. If status returns a value of 2, then an optimal solution has been found and the list of subtours is returned as well as terminating the algorithm. In contrast, if the status returned is not equal to 2, then the solution generated is not optimal and the model needs to be relaxed in such a way that the model produces a feasible solution. A feasible relaxation is an optimisation model that minimises the amount of which the linear constraints of the main model are violated [17]. In Gurobi, there are several options that one can choose from to perform the relaxation. For this optimisation model, we decided to split the relaxation part of the code into two parts; the first general relaxation and the second specified loop relaxation. The reasoning behind this was to make the code more adaptable as well as almost guaranteeing a solution. Moreover, the first relaxation takes less time as it is done more generally than the second loop relaxation which if lucky a valid solution will be generated in a smaller amount of time, but it is less likely that a correct solution will be generated. With this in mind, the best way to perform the first relaxation was to use "feasRelaxS(relaxobjtype, minrelax, vrelax, crelax)". Gurobi's "feasRelaxS()" provides a more generalised way of relaxing the constraint by modifying the model in such a way that a feasible relaxation is created. The first parameter is set to "0" which minimises the summation of the magnitudes of the bound and constraint violations [17]. In other words, if a constraint is violated by 1.0 when it should be 0.0, then it would contribute 1.0 to the feasibility relaxation objective [17]. As we deal with binary values, specifying the first parameter as "0" or "2" does not make much difference as "2" gets the total number of the bound and constraints violations which would still contribute to the same to the feasibility relaxation objective because of $1.0 + 1.0 + 1.0 = |1.0, 1.0, 1.0| = 3$. The integer values u are very minimal compared to the binary values x, which does not make drastic changes to the feasibility relaxation objective. Next, the second parameter is set to "False" which is the type of feasibility relaxation to perform [17]. By setting this parameter to "False", the returned model is optimised in such a way that the solution minimises the cost of the violation [17]. Although, setting this parameter to "True" can still produce a valid solution this can be very expensive it minimises the original objective and will take a lot of time. To continue, the third parameter is set to "False" to denote whether a variable boundary can be violated. As we always want all constraint bounds to be taken into consideration, defining this as "False" blocks such bounds from being violated. Finally, the last parameter is set to "True" to ensure that variables x and u are relaxed so that a new solution can be found. However, this can also relax the main constraints of the model, including the degree constraints of the nodes which can lead to an incorrect feasible solution. Therefore, once the "optimize()" method is called again each solution generated must be checked to ensure the results are as expected and if satisfied the algorithm is terminated.

If the solution after the first relaxation remains incorrect, then the model moves onto a more specified relaxation technique until a feasible solution is found. The code enters a while loop and before the relaxation is applied the main degree constraints are added again. As previously the "feasRelaxS()" relaxation technique has been applied to relax variables including the binary variable x responsible for the degree constraints, these need to be re-added so that no incorrect solutions are generated. If they are not added a solution that is expected to be $[[14, 6], [13, 4, 7], [12, 2], [11, 0], [10, 1], [9, 5], [8, 3]]$ could end up in an infinite loop as it would get stuck with a solution like $[[14], [6], [13], [4], [7], [12], [2], [11], [0], [10], [1], [9], [5], [8], [3]]$ which means that no node is connected to another node and therefore no subtours are generated. Without delay, the relaxation is applied. This time the model uses the "feasRelax(relaxobjtype,

minrelax, vars, lbpen, ubpen, constrs, rhspen)" method. As previously mentioned, the "relaxobjtype" remains set to "0" for the same reasons. On the contrary, the "minrelax" parameter is switched to "True" so that a more complicated relaxation is performed by attempting to minimise the initial objective of the model; the total cost travelled is minimal. In addition, the "vars" parameter is defined using only the variable x. The reason for doing this is to ensure that the work distribution remains fair and therefore the variable u is unbothered. The following two parameters are defined by taking the value "None" as "lbpen" and "ubpen" are the lower bound and upper bound penalties to be applied respectively for violating the defined variables "vars". Lastly, the "constrs" parameter takes the linear constraints that can be violated but as we do not want any constraints to be violated this is set to "None" alongside to "rhspen" which is also set to "None" because there is no penalty to be applied as no linear constraints will be violated. Once the relaxation is done, the degree constraints are added back again into the model before the "optimize()" method is called again. Just as before, each solution is checked and if a valid one has been generated the algorithm terminates and returns the assigned tours. It is must be noted that the while loop can be altered to having a maximum number of loop iterations to limit the time but with the possibility of an infeasible solution being generated and therefore no job assignment being done. At this instant, the algorithm has managed to generate a valid solution by minimising the main objective as well as ensuring all the constraints are met.

4.2.2 Database and Interface Development

The aim of this project was not only to create an algorithm that makes valid job assignments but also to try to make this into a more realistic product.

To do so, a database was created on the Cardiff University's School of Computer Science server. This database holds one table including the employees' information and one table holding the clients' information. The fields of the tables are an id, a first name, a last name, an optional email address, a phone number and an address including two separate fields that hold the latitude and longitude of the exact coordinates accordingly. For future use (that is not explored in this paper), the "Employees" table also holds an extra field storing the gender of each employee so that clients can request a male or a female helper to visit them. For demonstrating how this would work, dummy data has already been added in the database which does not relate to any real people.

The interface class is created using the kivy library as mention earlier. The program begins by attempting a connection with the Cardiff University's server by accessing the "getDBData()" method within the "useDatabase" class. If a connection is established, then the program retrieves all the necessary data from both databases; employees and clients. Otherwise, random data is generated. Once all the data has been retrieved or initialised, the main job assignment algorithm (section 4.2.1) is called by calling the "runAlgorithm" function within then "useDatabase" class. At this point in time, the assigned list of people's ids is returned and can now be used for the front end of this software. In the API code, an empty "TreeView" object is initialised. This object is later updated and filled with the names of both employees and clients that show how the assignment was done. This "TreeView" object is very similar to a drop-down menu using a more hierarchical approach. Next to this object, there is a map view showing how the locations of both employees and clients.

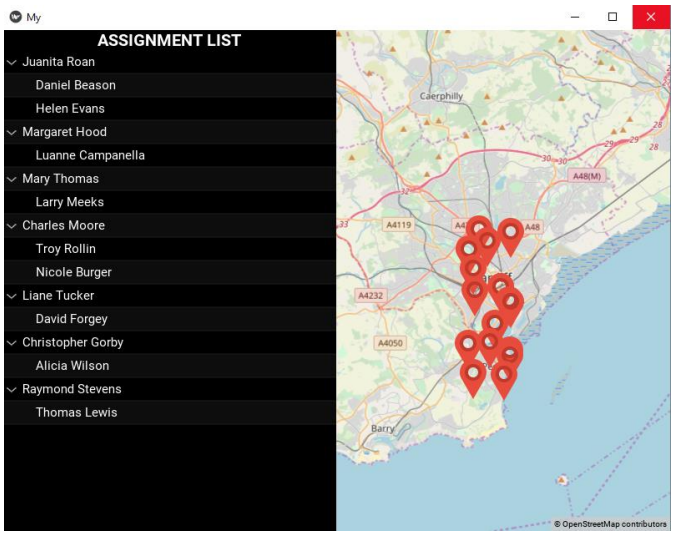

Figure 25-Expanded List, each name that can be expanded represents an employee and the leaf nodes represent clients

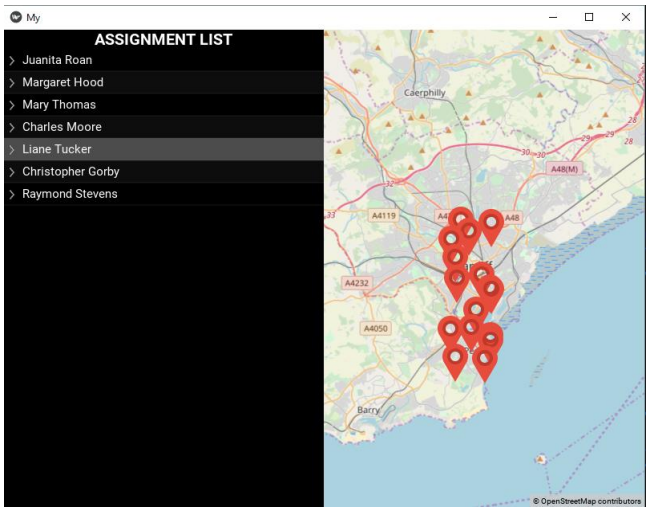

Figure 26-Collapsed List, each name that can be expanded represents an employee and the leaf nodes represent clients

5 RESULTS AND EVALUATION

5.1 TESTING AND RESULTS

5.1.1 Overview

All tests were carried out using an ASUS laptop running Windows 10 with an Intel Core i7 processor, RAM of 8 GB and four core processors. The algorithm was tested by running the code thousands of times while altering the number of employees and clients used. Please note that the code run did not attempt any unnecessary connection but only the pure original algorithm responsible for the job assignment. A few lines of code were used to save the outputs of the algorithm to a CSV file.

```
with open('testFile_RandomAssignment7.csv', 'w', newline='') as f:
  thewriter = csv.writer(f)

  thewriter.writerow(['execution No.', 'time','tour', 'error', 'relax', 'Employee No.', 'Patient No.'])
  for i in range(500):
    start = time.time()
    tour,e,n, relax = HHS.makeRandomAssignments()
    end = time.time()
    errorTour = ''
    if len(tour)!=e:
      errorTour = 'Error'
    thewriter.writerow([i, (end-start), tour, errorTour,relax, e, n])
f.close()
```

Figure 27-Test code used to save results of Job Assigning Algorithm to a CSV file

Moreover, as mentioned before, this paper only explores cases when $e \leq n$, where e is the number of employees and n is the number of clients. The reason why we cannot explore cases where $e > n$ is because of the way the degree constraints are set, we force all nodes to have one edge entering them and one edge leaving them. When $e > n$ some employee nodes must be left without any edges and the model we recommend will indicate these as incorrect solutions. Furthermore, if we were to test the code using $e = 5$ then the random number of clients was chosen such that $n \in \{e, \dots, r\}$ where r is the maximum number of clients. Assume that $r = 10$, then we could test all possible pairings such as $\{e = 5, n = 5\}, \{e = 5, n = 6\}, \{e = 5, n = 7\}, \{e = 5, n = 8\}, \{e = 5, n = 9\}, \{e = 5, n = 10\}$. This line of code "`n = random.randint(e, 10)`" was used to generate the random n every time the code was executed. Finally, be aware that the maximum number of employees and clients used was normally 10 because Gurobi uses the heuristic method of Branch and Bound (mentioned in section 4.1) which makes it a lot slower to test larger numbers.

5.1.2 Separated and Merged Constraints Testing

As mentioned previously (section 4), the algorithm uses some sort of relaxation technique to help relax the constraints in such a way that a feasible solution is achieved even if no optimal solution is found. Although this is not an ideal solution, it manages to almost guarantee that a solution is generated; even if that means a lot of time. We tested the algorithm using the separate constraints (11), (12) and (13) against the one merged constraint (8) mentioned in section 4.2.1. The following tables show the average time and the maximum time taken to run the job assignment algorithm 500 times as well as the success percentage with a maximum number of clients to be 10. We define a successful run to be a job assignment that has been solved immediately the first time or right after the first relaxation.

MERGED				SEPERATED			
EMPLOYEE NO.	AVERAGE TIME (s)	MAX TIME (s)	SUCCESS %	EMPLOYEE NO.	AVERAGE TIME (s)	MAX TIME (s)	SUCCESS %
1	0.117713066	0.772914648	100	1	0.140451313	1.115583658	100
2	0.177376003	2.472691536	100	2	0.198419548	2.467594862	100
3	0.159596399	1.184180498	91.8	3	0.17743153	3.87792778	90.4
4	0.169446337	0.921843052	74.8	4	0.219627829	2.359296322	74.8
5	0.364574856	2.613068819	71.4	5	0.417331936	3.78203845	84
6	0.488716842	5.179524899	80	6	1.820089893	32.02139783	80
7	3.281622921	32.57843685	71.2	7	1.29220625	12.13509536	69.8
8	11.94707751	146.8120389	86.2	8	2545.582845	12727.89875	80
9	0.209590359	1.2178092	100	9	0.210201814	0.550940752	100
10	0.206706193	1.225807428	100	10	0.25113149	0.590906143	100

Table 1-Timing results from running the optimisation algorithm using a random number of clients between 1 and 10 each time

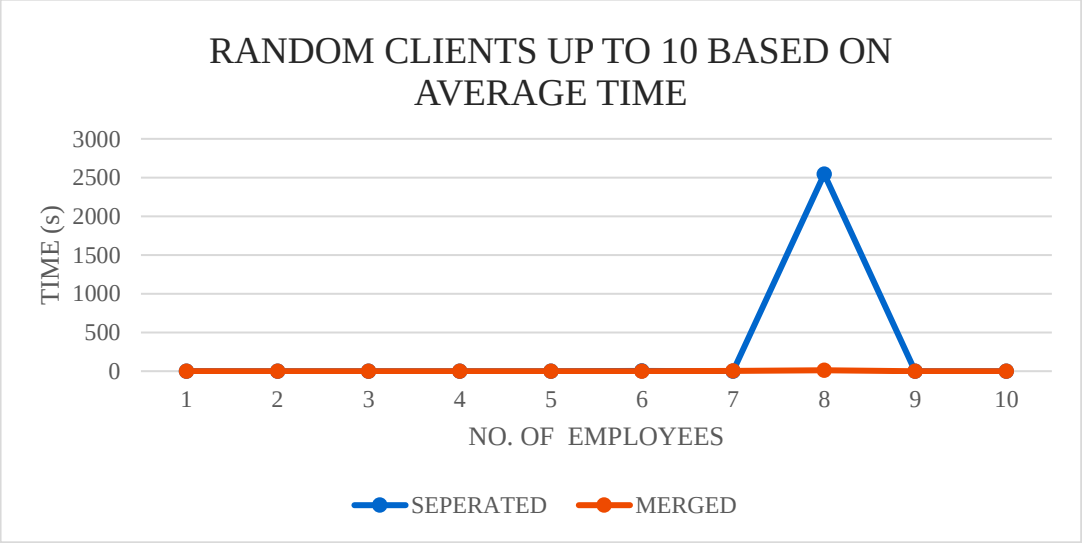

Figure 28-Graph comparing merged and separated constraints

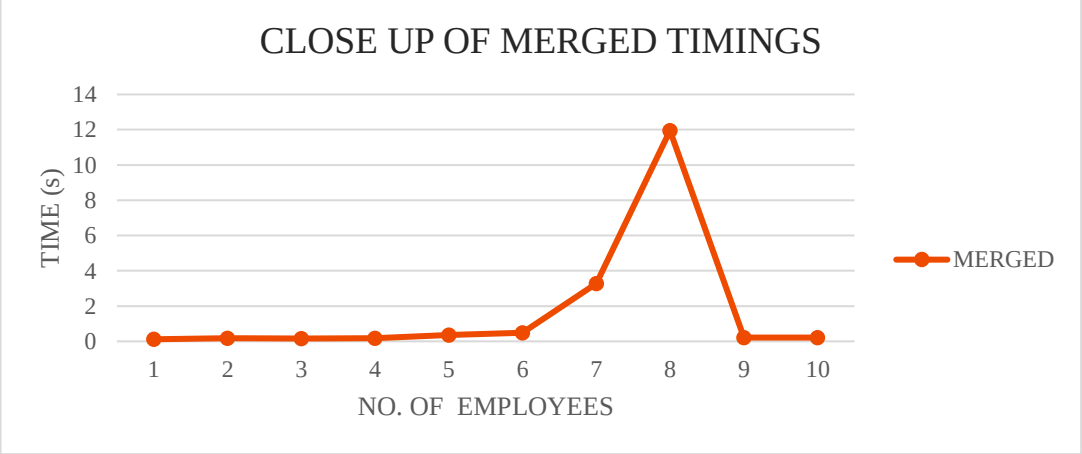

Figure 29-Close up of merged timings

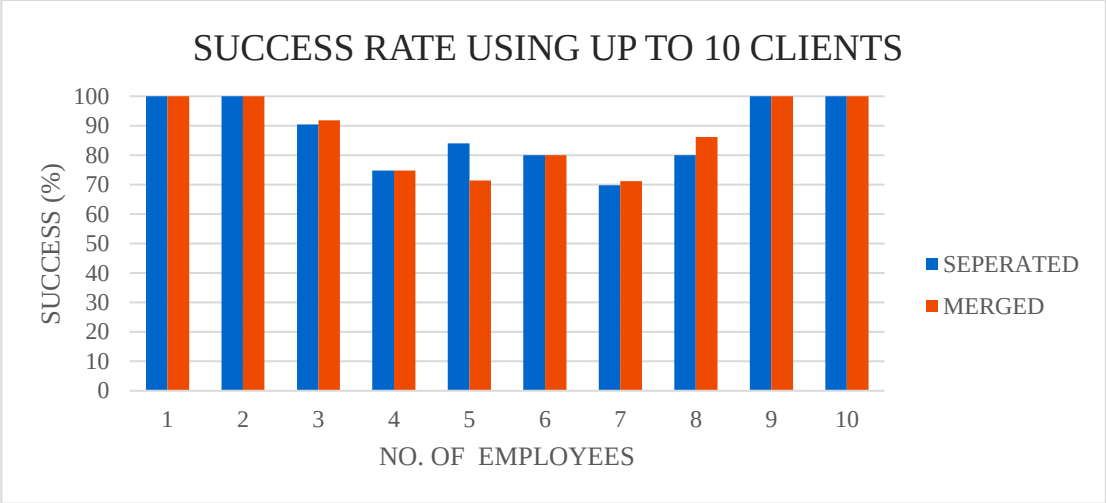

Figure 30-Success rate using between 1-10 random clients

We can clearly observe that although in most cases the average time is roughly the same, for highly complex cases such as having 8 employees and 10 clients having separated constraints makes the problem computationally harder at about 216 times slower. Given these points, the decision taken was to keep the merged version of the constraints (8) as it seems to provide better and more consistent results.

5.1.3 Employee and Client Range Testing

To explore the algorithm, even more, we compared the success rate when using up to 10 clients and employees as well as the run time for unsuccessful and successful executions to analyse how many cases were responsible for slowing down the algorithm.

		10 CLIENTS		
EMPLOYEE NO.	UNSUCCESSFUL TIME (s)	SUCCESSFUL TIME (s)	UNSUCCESSFUL NO	SUCCESSFUL NO
1		0.117713066		500
2		0.177376003		500
3	0.391671582	0.138866372	41	459
4	0.318786038	0.119134032	126	374
5	0.990753092	0.113752761	143	357
6	1.999260809	0.11108085	100	400
7	11.06523911	0.133193901	144	356
8	85.77666836	0.127491046	69	431
9		0.209590359		500
10		0.206706193		500

Table 2-Detailed table of successful and unsuccessful times using up to 10 clients and employees

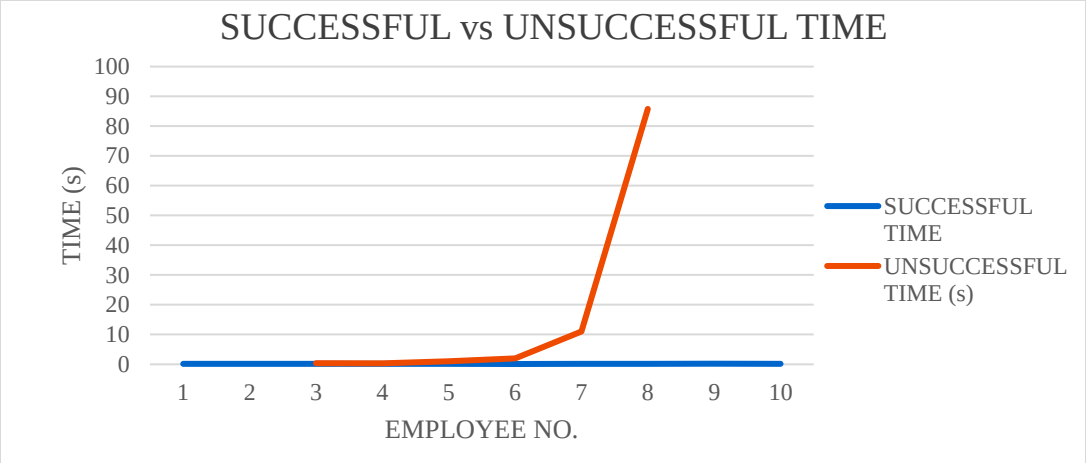

Figure 31- Successful VS Unsuccessful timings using up to 10 clients and employees

From the above results we can see that although the unsuccessful attempts are a lot less than the successful ones, they do take awfully a lot more time, especially when we use 8 employees and 10 clients. To explore this further, we tested the algorithm by using up to 6 employees and up to 20 patients. However, when trying to use $7 < e \leq r - 2$ employees where $r = 20$ the computational power required is extremely high, making it a lot more difficult to test such cases. The algorithm was run about 100 times for each number of employee and these were the results:

MERGED			
EMPLOYEE NO.	AVERAGE TIME (s)	MAX TIME (s)	SUCCESS (%)
1	1.887895392	55.6338582	100
2	6.544010937	149.1195877	100
3	0.589078774	5.082202435	80
4	2.035802148	98.02056432	56.56565657
5	2.773656948	27.3174026	44.03669725
6	14.07091473	67.05212283	31.57894737

Table 3- Testing with up to 6 employees using r=20

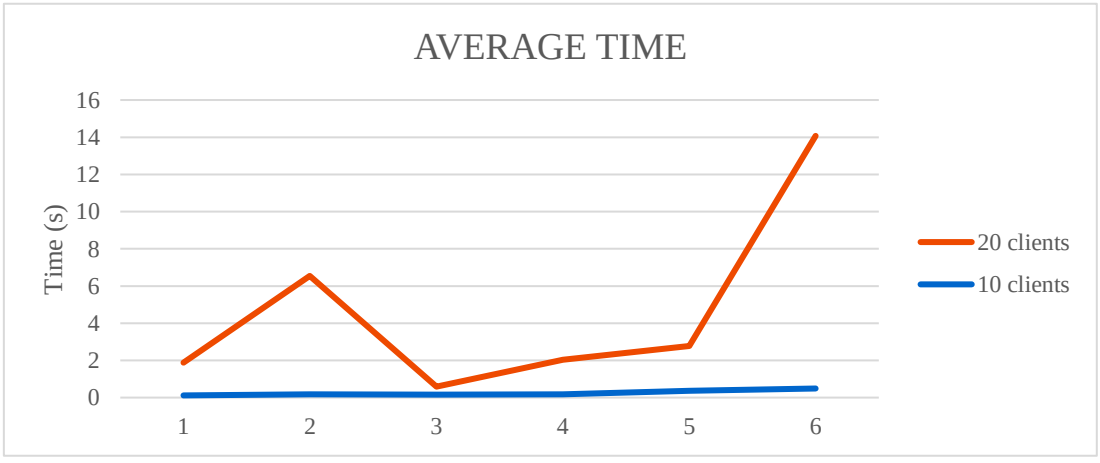

Figure 32- Comparing the average time when using up to 10 clients and up to 20 clients

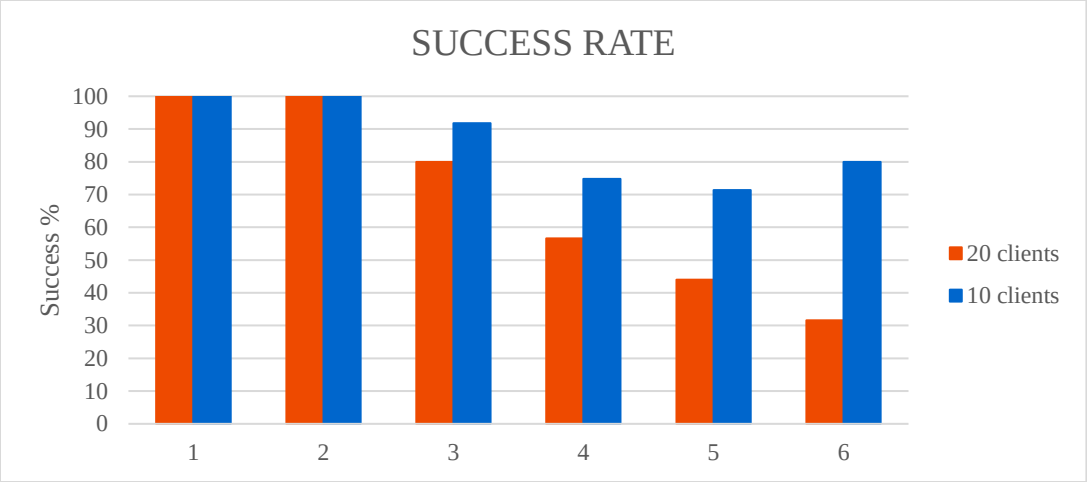

Figure 33- Comparing the success rate when using up to 10 clients and up to 20 clients

In conclusion, when using employees where $e \in \{3, \dots, r-2\}$ then the slower the algorithm gets. More specifically, the closer we get to $r-2$ employees the executed average time gets exponentially higher but as this could not be fully tested with a large range number of employees such assumptions are not 100% accurate. However, we can see from the above figures that this is not the case when using $e \in \{1,2\,r-1, r\}$ as the chance of the algorithm being completely successful by finding a solution either immediately or after the first relaxation is an outstanding 100%. To examine this further, we tested this with $r \leq 12$.

EMPLOYEE NO.	n	AVERAGE TIME (s)	MAX TIME (s)	SUCCESS (%)
1	9	0.25025756	3.13450861	100
2	9	0.393527513	2.210652828	100
8	9	0.203058455	2.213652134	100
9	9	0.233186067	1.013841629	100
1	10	0.117713066	0.772914648	100
2	10	0.177376003	2.472691536	100
9	10	0.209590359	1.2178092	100
10	10	0.206706193	1.225807428	100
1	11	0.15734372	2.043950796	100
2	11	0.164565022	1.750721931	100
10	11	0.150609552	0.528916359	100
11	11	0.154332888	2.367507219	100
1	12	0.462851535	6.280012846	100
2	12	1.437411141	42.92284346	100
11	12	0.226590777	1.126821756	100
12	12	0.259233869	1.570754051	100

Table 4- Testing Accuracy for e=1, e=2, e=r-1 and e=r

As expected, this gives 100% accuracy. At this point, we can notice a pattern that the more clients we use the slower the algorithm and the lower the success rate, this excludes using $e \in \{1,2\,r-1, r\}$ employees as the average time is relatively low and the success rate is relatively high if not 100%.

5.2 RESULTS EVALUATION

Overall, the algorithm always manages to give a solution and is subject to all the constraints. The assignment is completed correctly with the following deliverables being met; no more than one employee in each subtour, all employees are used and all patients are visited, the total distance travelled is minimised as well as keeping a fair workload for all employees. As demonstrated above, the algorithm works better with a lower number of employees being about 5 and a fairly low number of employees at about 20. The largest test run on the machine was $e = 5$ and $n = 50$ that took 6423.262 seconds which is more than 1 and a half hours. As we presumed, the algorithm depends strictly on the locations of the employees' houses in respect to the clients' houses which can cause the algorithm to take longer periods of time to execute. When locations were scattered in a way where relationships are fairly easy to indicate, assignment happened relatively fast. However, when locations had no correlation, relationships between vertices was difficult to indicate which took longer time to complete. For example, when using 3 employees and 6 patients if each employee has 2 patients close to them this was an easy assignment. Otherwise, if one employee has all 6 patients close to them and the other 2 employees were far, assignment was more difficult to complete hence more time. Nonetheless, this was not always the case as some set of locations abstractly scattered across the surface would generate a solution relatively fast but others differentiating slight would take minutes or even hours. The reason for this, is the nature of computational optimisation as it tries to find intersecting points which is sometimes difficult to achieve within the given constraint area.

It is important to note that this is not entirely the algorithm's fault. Due to its high complexity, it requires an extreme amount of time and computational power to use a large number of employees or clients which made it impossible for the machine that was used to run such tests. Ideally, a better and more powerful machine should be used in the future to perform more tests to help understand better when the algorithm performs slow including specific employee to client ratio comparisons.

6 FUTURE WORK

Some tasks initially set were not achieved due to the lack of appreciation for the complexity of this project within the time period provided. The base problem explored in this paper turned out to be more difficult than anticipated. Time restriction made it impossible to expand this algorithm into an even more complex and intelligent solution by having extra constraint specification as well as a more intuitive user interface.

6.1 ALGORITHM DEVELOPMENT

6.1.1 Better Runtime Execution

This project required a lot of research and a lot of programming trials. The bigger issue has been resolved by finding and constructing the correct main constraints as well as the fitting lazy constraints that are used to deal with inequalities that may arise. As mentioned previously in section 4, solutions that are violated for a second time are still being explored by adding the main degree constraints again, relaxing the model and then adding those constraints back into the model until a feasible solution is generated. Nevertheless, this is not an ideal implementation as it has a very big disadvantage; the execution time. As the number of employees and/or clients becomes larger the time for the model to generate a solution becomes exponentially higher. A big company with a lot of staff that provide home-help services to many areas might have to wait a lot of time to make the assignments and will need very powerful machines to complete such computations.

A computational optimisation problem can be solved in many ways. Although a solution is generated most times, deeper research and even potentially alteration of the constraints in a certain way could lead to a more reliable optimisation model. In addition, the model can be expanded to allow solutions where $e > n$.

Another potential solution to this is to split up the employees and the clients using some sort of k-means clustering. The k-means clustering will take a large set of employees and clients and will return k clusters. This way the computational optimisation model can then be applied at each generated cluster.

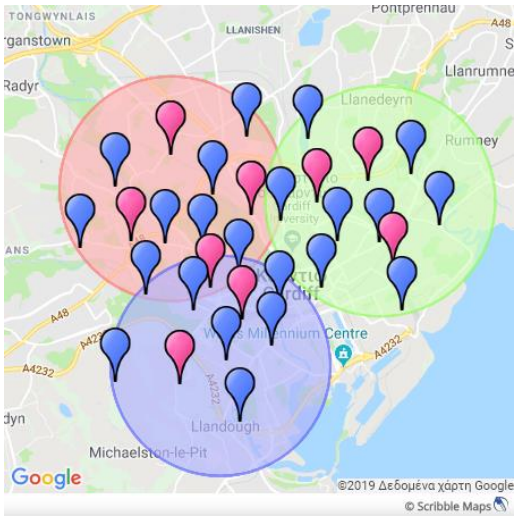

Figure 34 - Potential example of clustering created using an online tool [18]

The above Figure 34 demonstrates how a complex problem can be simplified. The red, green and blue circles are the generated k clusters by applying some kind of clustering method. The clustering to be used could be some agglomerative method which begins by having all employees and clients as separate entities and then iteratively merge entities together into a cluster by getting the closest ones until a satisfying number of clusters is created or until each cluster includes a satisfying number of entities. Each circle represents an individual subproblem that can later be solved using the algorithm explored in this paper.

6.1.2 Additional Algorithm Features

Once the algorithm is working perfectly and a lot faster, more constraint options can be added to help make even more complex assignments. Such constraints evolve some more personal requirements the employees or clients might have depending on the flexibility of the company such as; a client might only want to be visited by female workers or an employee can only work between $07:00 - 11:00$.

To implement the gender preferability, there could be an additional dictionary which holds key-value pairs where the key indicates the employee and the value is a binary where 0 indicates a male and 1 a female. For instance, if there is a set of employees $E = [0,1,2,3,4,5]$ then a dictionary indicating the gender would be $genderDict = \{0:0, 1:0, 2:1, 3:0, 4:1, 5:0\}$ which shows that employees $\{0,1,3,5\}$ are male and employees $\{2,4\}$ are female. Identically, an additional dictionary needs to be introduced to denote whether a client has any preferences on the gender of the helper. Let's assume $N = [6,7,8,9]$ then $clientGenderDict = \{6:1, 7:0, 8:1, 9:-1\}$ which shows that clients $\{6,8\}$ would prefer being visited by a female, client $\{7\}$ would prefer a male and client $\{9\}$ does not have any preferences. With a similar concept in mind, we can implement the working times also. In this case, the value within the dictionary can be a list of binaries where each indexed value represents the time period. Assume a set of employees $E = [0,1,2,3,4,5]$ then the dictionary would be $timeDict = \{0:[0,1,0], 1:[1,1,0], 2:[1,0,0], 3:[0,0,1], 4:[1,1,1], 5:[1,0,1]\}$. The key indicates the employee and the list indicates when the employee can work. The list holds three values where the first value denotes that the employee can work between $7:00$ and $11:00$, the second value shows that the employee can work between $11:00$ and $15:00$ and the last value represents working between $15:00$ and $19:00$. If the indexed value is 1 then the worker can do the specified shift, otherwise, they cannot work during that period. For example, employee 0 can only work between $11:00$ and $15:00$ where employee 4 can work all three shifts and so on. Later, some additional constraints can be added where certain nodes cannot exist in specified subtours.

6.2 FURTHER INTERFACE DEVELOPMENT

To make this project even more realistic and into a functional product that can later be sold to companies, a better interface could be implemented. An interface prototype was created to demonstrate a possible User Interface that the software can use. The following screens were created using the prototyping tool "JustInMind" [19] which allows easy and fast development of prototypes.

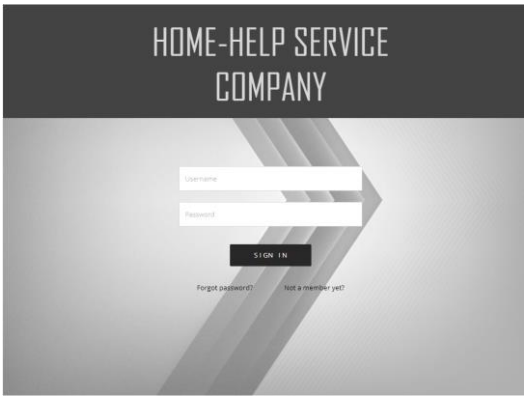

Figure 35-Login Screen

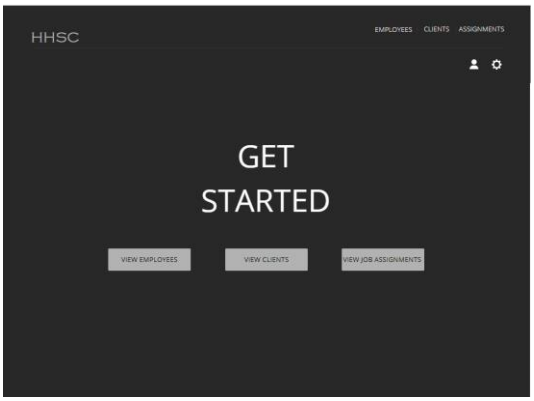

Figure 36- Main Screen

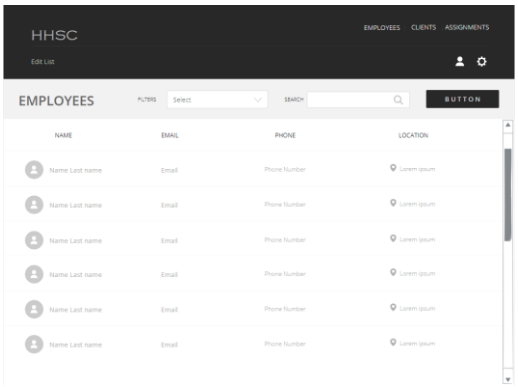

Figure 37- View Employees

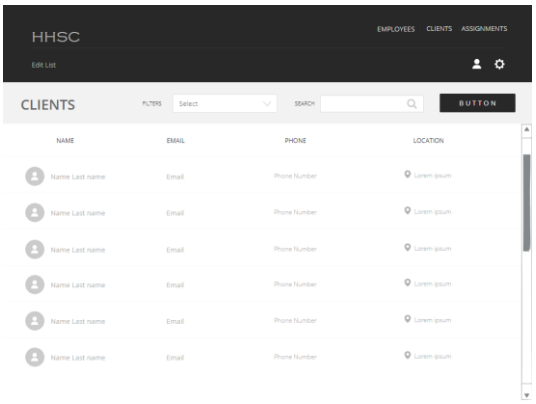

Figure 38- View Clients

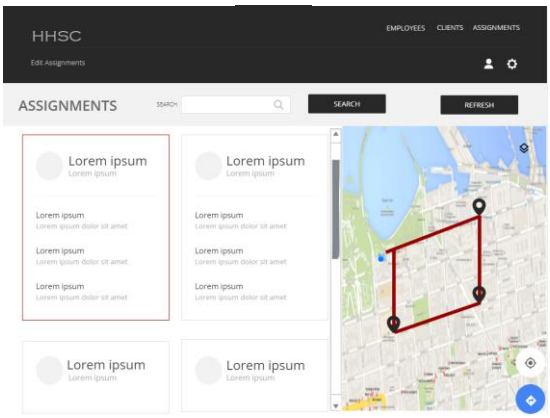

Figure 39- View Job assignments

The first two screens, Figure 35 and Figure 36 show the login screen and the first screen the user encounters after login is successful respectively. In “View Employees” screen (Figure 37) and “View Clients” screen (Figure 38) the user can view all details associated with an employee or client respectively. Users can also edit, delete or add entities and the database will automatically be updated. In addition, the “View Job Assignments” screen (Figure 39) makes it easy for the company to manage the assignments between its staff and patients. A “Refresh” button is available to trigger the optimisation algorithm and recalculate the assignments. Each box holds the name of the employee and the names of clients to be visited. When a box is selected then the route to be taken is shown on the map on the right-hand side. Users can again edit, delete or add assignments.

7 CONCLUSIONS

This project has explored what we consider to be one of the first (if not the very first) studies of a more complex variation of the MDMTSP where all depots are used using complex parts of the computational optimisation and integer linear programming world. Although mathematical models do exist, very few computational models are available. We have managed to present a reasonable solution and a working model for any number of employees e and clients n as long as $e \leq n$. The problem itself is NP-hard as it cannot be solved in polynomial time and is equivalent to the Travelling Salesman Problem when only one employee is used as all instances of TSP can be transformed to an instance of this problem explored, $TSP \propto problem\_explored$. Nonetheless, large numbers of employees and/or clients require a lot of computational power, which makes the job assignments a lot slower. The reason behind this, is the heuristic methodology of branching and bounding and its recursive nature, as it tries to explore different solutions until one is found.

Optimisation models are getting more popular and a lot of time is spent into researching ways to utilise them even further. We believe that optimisation techniques are the way into the future as they can help automate a lot of procedures with minimal effort.

8 REFLECTION ON LEARNING

This project has been an amazing journey. I am proud to have explored such interesting areas of the mathematical optimisation world and give back to the community by developing and implementing the model discussed in this paper. To see how long ago the first optimisation technique and the first problem was introduced to still being explored today is fascinating to me. We as scientists have come far by resolving these problems and even further introducing and exploring more complex ones.

Nonetheless, this has not been an easy journey. Optimisation techniques and Integer Linear Programming was foreign to me and taking this project was a challenge on its own. I have spent an incredible amount of time exploring other similar problems such as the TSP (section 2.1), the mTSP (section 2.2) and the MDMTSP (section 2.3) but even the Classical Vehicle Routing Problem (not explored in this paper) which ended up not being of any help. This shows how crucial it is to detect which papers and problems explored by other scientists can be of help and rejecting those that are not useful. In due time, I realised that the more I understood the problem the better I could distinguish what I could use to explore this and what not. Even after finding helpful documents and papers on similar issues, it took a lot of trial and error until a correct model was developed which was not an easy attempt. The lack of documentation regarding similar variations was minimal but close to none specifically on the variation explored in this project which made this task even harder to complete. The major challenge was not only to come up with the mathematical constraints but also to make them work in the model efficiently. An obstacle that I had to overcome was finding the balance between the mathematics and the programming behind this model as a lot of the times the code would not perform as expected.

As a result, I have gained an incredible set of skills that can be used in the future. I have developed better research skills and expanded even further my programming skills by learning how to use Gurobi as well as a brief overview of kivy which were both unfamiliar to me. I have also grown to appreciate the computational optimisation world as it is widely explored by mathematicians and computer scientists daily. I am confident that I have gained a deep understanding of this era and will continue to keep up to date with future developments. Looking forward to seeing how far the computational optimisation world can go.

REFERENCES

[1] W. T. F. Encyclopedia, “Mathematical optimization,” *Wikimedia Foundation Inc.* [Online]. Available: https://en.wikipedia.org/wiki/Mathematical_optimization. [Accessed: 02-Apr-2019].

[2] Wikipedia: The Free Encyclopedia, “Linear programming,” *Wikimedia Foundation Inc.* [Online]. Available: https://en.wikipedia.org/wiki/Linear_programming. [Accessed: 02-Apr-2019].

[3] Wikipedia: The Free Encyclopedia, “Travelling salesman problem,” *Wikimedia Foundation Inc.* [Online]. Available: https://en.wikipedia.org/wiki/Travelling_salesman_problem#History. [Accessed: 31-Mar-2019].

[4] “The Travelling Salesman Problem with Integer Programming and Gurobi.” [Online]. Available: http://examples.gurobi.com/traveling-salesman-problem/#demo. [Accessed: 07-Feb-2019].

[5] “Multiple Traveling Salesman Problem (mTSP) | NEOS.” [Online]. Available: https://neos-guide.org/content/multiple-traveling-salesman-problem-mtsp. [Accessed: 21-Feb-2019].

[6] X. Xu, H. Yuan, M. Liptrott, and M. Trovati, “Two phase heuristic algorithm for the multiple-travelling salesman problem,” *Soft Comput.*, vol. 22, no. 19, pp. 6567–6581, Oct. 2018.

[7] E. Benavent and A. Martínez, “A polyhedral study of the Multi-Depot Multiple TSP,” no. Parragh 2010, pp. 1–34.

[8] “Gurobi Optimization - The State-of-the-Art Mathematical Programming Solver.” [Online]. Available: http://www.gurobi.com/index. [Accessed: 18-Apr-2019].

[9] “Mixed-Integer Programming (MIP) Basics | Gurobi.” [Online]. Available: http://www.gurobi.com/resources/getting-started/mip-basics. [Accessed: 19-Apr-2019].

[10] “MySQL :: MySQL Workbench.” [Online]. Available: https://www.mysql.com/products/workbench/. [Accessed: 20-Apr-2019].

[11] “Kivy: Cross-platform Python Framework for NUI Development.” [Online]. Available: https://kivy.org/#home. [Accessed: 10-May-2019].

[12] “draw.io.” [Online]. Available: https://www.draw.io/. [Accessed: 10-May-2019].

[13] “Developer Guide | Distance Matrix API | Google Developers.” [Online]. Available: https://developers.google.com/maps/documentation/distance-matrix/intro. [Accessed: 10-May-2019].

[14] W. T. F. Encyclopedia, “Cutting-plane method,” *Wikimedia Foundation Inc.* [Online]. Available: https://en.wikipedia.org/wiki/Cutting-plane_method. [Accessed: 18-Apr-2019].

[15] “Cuts.” [Online]. Available: http://www.gurobi.com/documentation/8.0/refman/cuts.html.

[Accessed: 09-May-2019].

[16] “Optimization Status Codes.” [Online]. Available: https://www.gurobi.com/documentation/8.1/refman/optimization_status_codes.html. [Accessed: 18-Apr-2019].

[17] “Model.feasRelaxS().” [Online]. Available: http://www.gurobi.com/documentation/8.1/refman/py_model_feasrelaxs.html. [Accessed: 18-Apr-2019].

[18] “Create Maps : Scribble Maps.” [Online]. Available: https://www.scribblemaps.com/create/#/lat=36.879620605027014&lng=-40.78125&z=3&t=hybrid. [Accessed: 10-May-2019].

[19] “Free prototyping tool for web & mobile apps - Justinmind.” [Online]. Available: https://www.justinmind.com/. [Accessed: 10-May-2019].