UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

SEPTEMBER 4, 2017

RESOURCE SCHEDULING ALGORITHM FOR MAINTENANCE PLANNING

INTERIM PROJECT REPORT SUBMITTED IN PARTIAL FULFILLMENT FOR THE REQUIREMENTS OF THE MODULE

BPJ420

KIRSTEN YOUNG

14062471

University of Pretoria

ANTIPLAGIARISM PAGE

I, Kirsten Young, student number 14062471 hereby declare that this report is my own original work, and that the references listed provides a comprehensive list of all sources cited or quoted in this report.

EXECUTIVE SUMMARY

"Company XYZ" is a company which outsources maintenance to various enterprises all over South Africa. Technicians are hired to travel to their customers which are geographically far from one another to perform maintenance on electrical devices such as servers, computers and air conditioners. An employee's workday consists of both their travel time and working time and so routing must be carefully considered in order to reduce travel costs. Company XYZ's employees find that their workloads are unbalanced i.e. some days they will work much longer hours than others. This has led to Company XYZ requiring a way to efficiently schedule their employees so that customers demand can be met, while keeping costs low, resource utilization high and workloads balanced. Fourier-E attempted solving Company XYZ's problem by creating a linear programming resource allocation model. The model worked but there is still much room for improvement. All the data was therefore already available in a device database which could be used in the development of a new solution.

After performing a literature study it was found that the problem at hand has many similar aspects to that of a Multiple Travelling Salesman Problem and so the many methods of solving this kind of problem were researched. The genetic algorithm was selected as the most suitable algorithm for solving the problem because of its short running time and the student's ability to code it. Specific selection, crossover and mutation techniques were used to evolve the initial population of solutions. With every new generation, a better schedule was found. The best solution of the final generation was selected as the schedule to analyse.

The genetic algorithm exhibited many advantages over using the existing linear programming method. The chosen schedule significantly reduced overtime, reduced travel distances and balanced resource workloads. It is up to the company to decide whether they should implement it or not. Company XYZ should validate the final schedule by using a testing team to ensure that the assumptions on which the model was based are acceptable.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

1. INTRODUCTION AND BACKGROUND

1.1. THE SERVICE INDUSTRY

There has been a rapid growth in the service industry over the last decade. This is due to the fact that companies are recognizing the benefits of outsourcing activities that are not part on their core functions (Linton, 2012). Outsourcing services such as accounting, IT or maintenance allows a company to concentrate on the activities that will make them money. Companies that provide a service work in very different way to manufacturing companies as people are the essence and core of the business (Dorne, 2008). Employees working in services are usually not confined to a particular facility; rather they travel to provide their services across geographical areas. Unlike the manufacturing industry, services cannot build up an inventory during times where there is less demand – it needs to be met as it is required. This places a great importance on service management as well as the logistics of having the right staff members at the right place and at the right time in order to satisfy customer demand.

1.2. COMPANY XYZ

The company for which this project will be undertaken would like to remain confidential and so for the purposes of this project, it will be referred to as 'Company XYZ.' Company XYZ is a national maintenance company to which numerous enterprises all over South Africa outsource their maintenance.

Employees are hired to travel to customers to perform scheduled maintenance on a wide range of electrical devices such as air conditioners, computers, telecoms, servers etc. Scheduled maintenance involves planning to maintain devices or machines at regular, predetermined intervals. This includes activities such as inspections, adjustments, regular services and planned shutdowns (O'Brien, 2017) in order to prevent machines from breaking down.

1.3. PROBLEM BACKGROUND

As can be seen in **Figure 1**, employees (resources) {R1; R2; …; RN} travel from their branches {B1; B2; …; BN} to customers' premises {PR1; PR2; …; PRN} where the electrical devices {ED1; ED2; …; EDN} on which they are required to work are kept. Each electrical device has its own service time. In a single day employees may be required to travel to multiple locations to perform services. A workday of an employee will thus consist of their travel time plus the time they spend working on electrical devices.

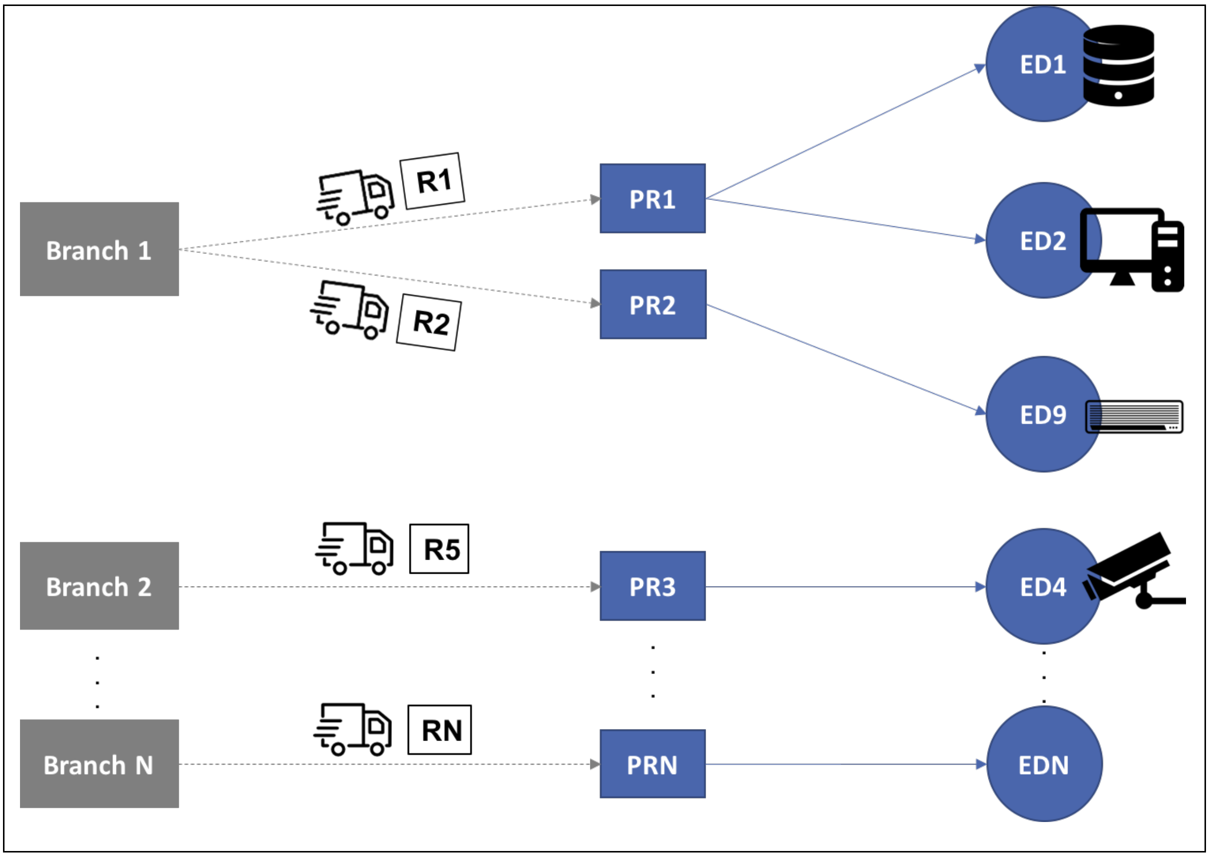


FIGURE 1: AN ILLUSTRATION OF THE HIGH LEVEL PROCESS COMPANY XYZ USES TO SATISFY THEIR CUSTOMERS

Customer's premises are geographically far from another and therefore routes need to be carefully considered so that employees can maximize their time spent working and reduce travel costs and overtime. A full-time employee should ideally work 40 hours per week (8 hours per day), but this is rarely the case. In one week an employee may work 50 hours and will have to be paid overtime. In the next week he may only work 30 hours, but will still have to be paid for a full 40 hours' worth of work because he was available. The total workload or the resource utilisation of Company XYZ is therefore unevenly distributed, as demonstrated in the figure below:

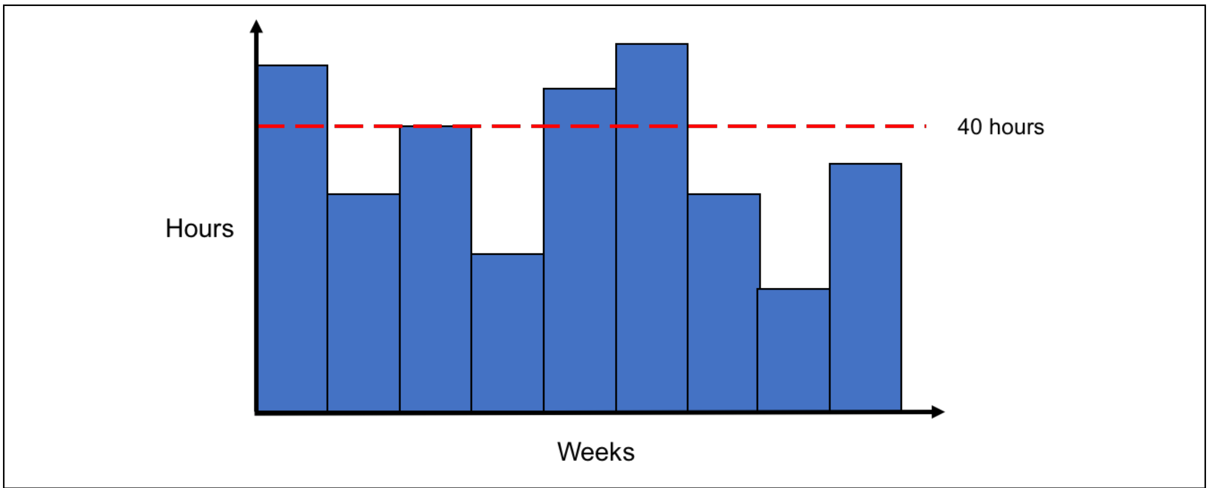


FIGURE 2: DEMONSTRATION OF UNEVEN WORKLOAD DISTRIBUTION

1.4. Fourier-E

Fourier-E (part of Fourier Approach) is an Industrial Engineering consulting company specialising in the areas of Decision Support, Cash Management, Operational Design and Supply Chain Engineering. Company XZY has been a client of Fourier's for many years. In 2009 Fourier was hired to help Company XYZ in the allocating of maintenance tasks in order to smooth out resource utilisation, and then once again in 2012 to build a pilot model for one region in which they operated which minimised travel in addition to allocating maintenance tasks to smooth resource utilisation. In 2016 this model was executed across all maintenance tasks and resources over all regions in South Africa. Fourier-E will therefore be very useful in providing data for this project.

2. Problem statement

Company XYZ has the logistical problem of deciding which jobs need to done, when they need to be done, at which customer premises and by which employees. Company XYZ requires a system which plans staffing in such a way as to balance the workloads of their employees, reduce costs and meet customer demand.

3. Project Aim/ Rationale

The aim of this project is to create a maintenance plan for Company XYZ which indicates which resources will be servicing which devices on which days. As seen in **Figure 3** below, this plan should optimize the workload distribution of employees as well as the routes they travel each day so that customers can be serviced in the most efficient way possible. This will lead to the improvement of customer satisfaction and the reduction of workforce costs (Dorne, 2008) such as overtime and idle time, as well as fuel and vehicle maintenance costs.

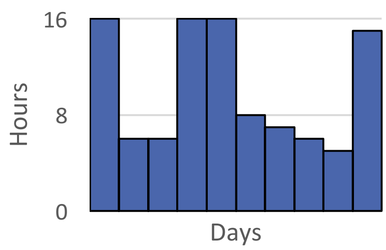
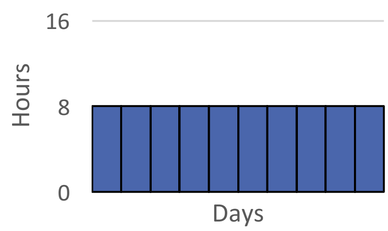
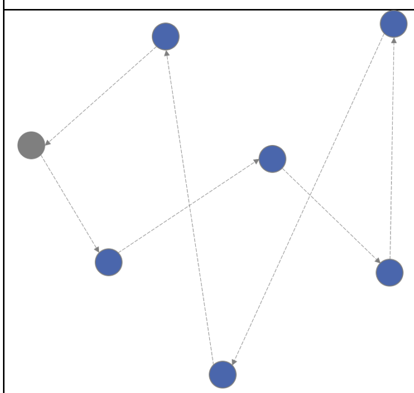
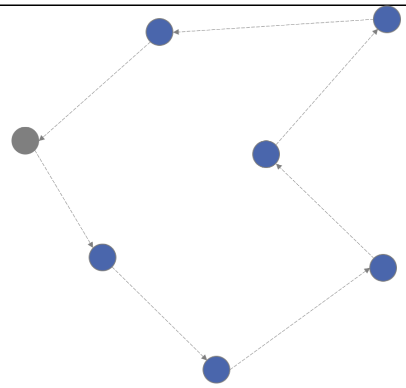
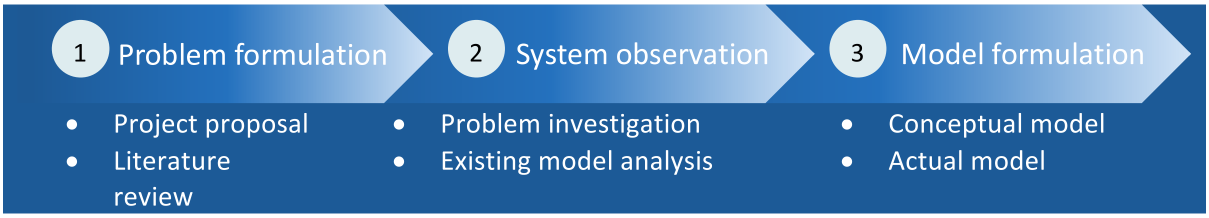
Current Situation		Ideal Situation
 	Moving towards	
Employees are working a different number of hours each day and some employees are working more hours than others.		Employees are scheduled in such a way that their days are more or less 8 hours and each employee has roughly the same amount of work to do.
	Moving towards	
Current routes are complex and not much thought has gone into them. Employees find themselves going back past places they have already visited.		Routes have been optimized to reduce travel time per day and thus travel costs so that employees have more time to perform their value adding jobs.

Figure 3: Current situation at Company XYZ versus what a maintenance plan will help achieve

4. Project Approach, Scope and Deliverables

This section provides the general approach that will be taken in the solving of Company XYZ's problem. The problem that needs to be addressed and the solution that will presented is typical of an Operations Research based project. T.A Taha (1976) defines Operations Research as "*…a scientific knowledge through interdisciplinary team effort for the purpose of determining the best utilization of limited resources*," which is what this project ultimately aims to do. According to Winston (2003) the following seven-step model-building procedure should therefore be followed as an approach to solving this Operations Research based problem:



Phase 1: Formulate the Problem

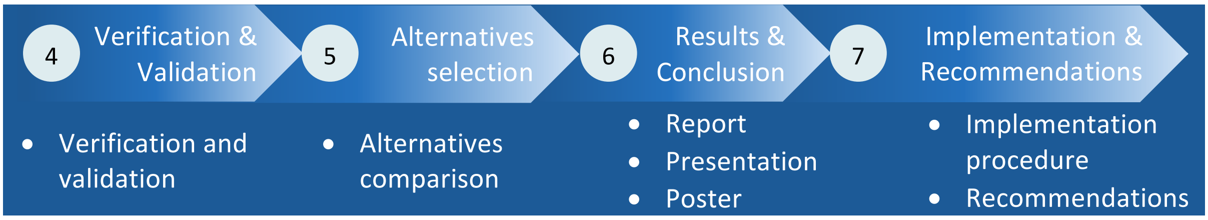
This phase will involve speaking to the employees from Fourier-E which were involved in the Resource Allocation Projects and having them explain the background of the problem. Individual research will be conducted to gain enough information about the project so that a project proposal can be completed. An in-depth literature study (**Section 5**) will then be undertaken to identify what kind of problem is being dealt with as well as what methods are available to solve it.

Phase 2: Observe the System

As mentioned in **Section 1**, Fourier-E has performed several projects at Company XYZ and thus have built up a large database of information pertaining to the operations of the company. This database will be used as the starting point for understanding how the company works and how all the different aspects of the company fit together i.e. regions, bases, customer premises, resources, equipment etc. A more in-depth discussion of the problem will then be presented. As part of the problem investigation (**Section 6**), Fourier's existing model will be analysed and the strengths and shortcomings of it will be discussed.

Phase 3: Formulate a Mathematical Model of the Problem

A conceptual model will be designed to show which inputs and outputs will be required, and the general way in which the model should work will be explained. The final model will then be developed using knowledge and background gained from literature studies and Fourier's model. The development of the final model can be found in **Section 7.**



Phase 4: Verify and Validate the Model

The actions that Company XYZ should take in order to verify and validate the completed model will be explained (**Section 8**).

Phase 5: Select a Suitable Alternative

Different types of scheduling algorithms and linear programming methods will be researched and compared with one other in order to determine if any are suitable for the application at Company XYZ or if something completely new needs to be formulated. These comparisons can be found in the literature review (**Section 5**).

Phase 6: Present Results and Conclusion

A final report, poster, and PowerPoint presentation will be created to present the final proposed maintenance plan as well as how it was developed.

Phase 7: Implement and Evaluate Recommendations

The final report will include recommendations on how Company XYZ should go about the implementation phase if they were to accept the proposed maintenance plan (**Section 9**).

PROJECT SCOPE

There is a large amount of data available to use. There are 6 regions in South Africa in which Company XYZ operates and so it would be a good idea to model only the Gauteng region initially. If the final algorithm works effectively, it can then be easily adapted to solve for other regions in the future. The Gauteng region has only one branch, which employs 36 full-time technicians. The companies which this branch services are shown in the map alongside.

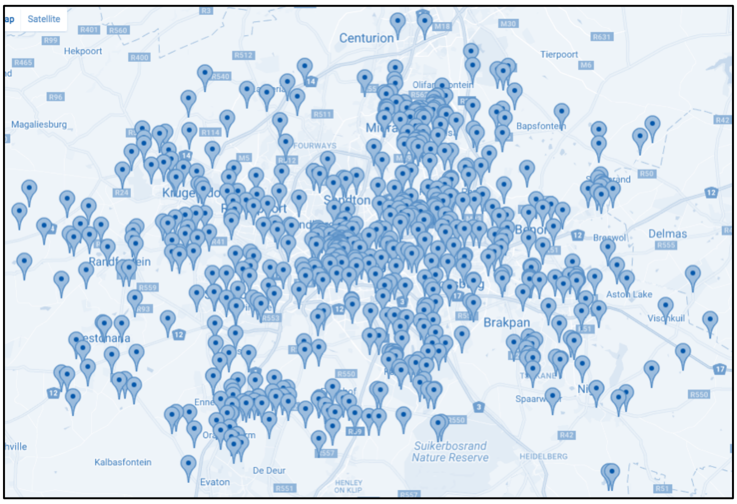


FIGURE 4: MAP SHOWING THE LOCATIONS OF COMPANY XYZ'S CUSTOMERS IN THE GAUTENG REGION

5. LITERATURE REVIEW

In this section a literature review is conducted to find and understand theory that could be applied to the problem at hand as well as specific applications of this theory about which researchers have written papers. There are a number of methods discussed below. One method or a combination of these different methods will be selected to solve Company XYZ’s problem, based on their suitability to the problem and the student’s abilities.

The problem at hand has elements of both scheduling and routing because tasks and technicians need to be scheduled in such a way as to complete all the tasks requested by customers, in addition to determining efficient routes which minimise the costs associated with travelling to geographically distributed customers.

Routing and scheduling problems can be presented as graphical networks (Haksever *et al.*, 2000). In **Figure 5** below, the circles are called nodes which represent pickup and/ or delivery points. Node 1 represents the depot or home base of the vehicles. The lines connecting the nodes are called arcs which can represent the time, distance or cost required to travel from one node to another. The goal of routing and scheduling problems is usually to minimize this time, distance or cost of travelling.

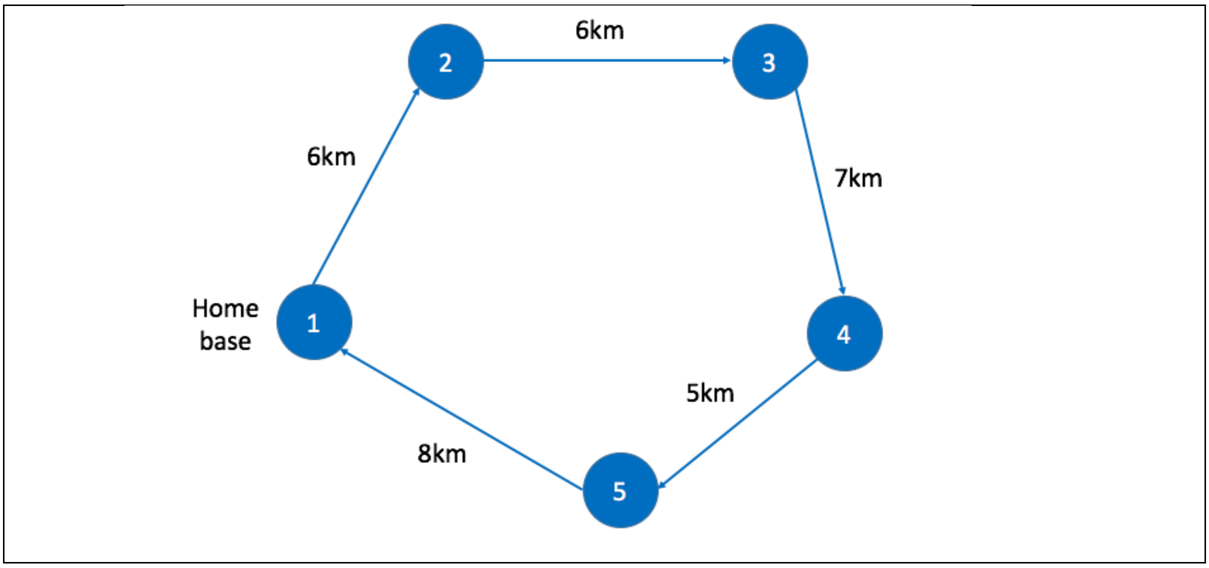


FIGURE 5: REPRESENTATION OF A ROUTING PROBLEM

Numerous types of routing and scheduling problems are mentioned in literature, each having a variety of ways in which they can be solved. The classification of a routing and scheduling problem depends on characteristics such as size of the delivery fleet, where the fleet is housed, capacities of the vehicles, as well as routing and scheduling objectives (Haksever *et al.*, 2000).

In the simplest and most well-known case of routing problems, the **Travelling Salesman Problem** (TSP), a single vehicle is required to visit a set of nodes. The output is a route which begins and ends at the depot which enables the vehicle to visit each node exactly once while minimizing travelling distance. In the case of the **Multiple Travelling Salesmen Problem** (mTSP), a fleet of vehicles is required to visit a set of nodes and each vehicle starts and ends at the depot. The objective is to determine a route for each salesman such that the total distance is minimized and that each city is visited exactly once by only one salesman. The **Vehicle Routing Problem** (VRP) expands on the multiple traveling salesman problem. In this case the capacity of the vehicles is restricted and the nodes to be visited have varying demands. In the **Chinese Postman Problem** (CPP), the demand for service occurs on the arcs rather that at the nodes, for example street sweeping and newspaper delivery.

From the descriptions of the cases above, it appears that the Multiple Travelling Salesman Problem most closely resembles the problem with which Company XYZ is faced, because there are multiple technicians but the capacities of their vehicles are not limited and the demand for their service occurs only at the nodes. Approaches for solving the mTSP above can be divided into two categories: exact algorithms and heuristic algorithms.

5.1 Exact Methods

The following traditional techniques have been used to find exact solutions for small problems but as the size of the solution space increases, so does the running time of these algorithms (Laporte, 1992).

5.1.1 Mathematical Programming

There exists a wide variety of mathematical programming methods, the most basic one being Linear programming (LP). This is the optimisation of a problem in which the objective and constraints are linear. In integer programming (IP), variables are restricted to be integers. When some but not all variables are restricted to be integers, it is a mixed integer program (MIP) (Pinedo, 2012). In binary integer programming (BIP), each variable can only take on the value of 0 or 1. Various algorithms exist to solve the TSP and variations of it. Some include the Branch-and-price algorithm and the Branch-and-bound algorithm.

5.1.2 Dynamic Programming

Dynamic Programming is very different to linear programming in that there is no standard mathematical formulation of the problem. It is "*a method that in general solves optimization problems that involve making a sequence of decisions by determining, for each decision, sub problems that can* *be solved in like fashion, such that an optimal solution of the original problem can be found from optimal solutions of sub problems*" (Lew and Mauch, 2007). Dynamic programming has often been used in the solving of Travelling Salesman Problems (Laporte, 1992) using the Held-Karp algorithm.

5.2 Heuristic Methods

When the number of variables gets too large, computers have difficulty in finding an exact solution. In the case of the TSP or variants of it, the difficulties in solving this problem arise from the large number of possible tours: the factorial of the number of cities to be visited (Larranaga *et al.*, 1999). In this case, heuristics methods should be used to find good, if not optimal, solutions to the problem (Winston, 2003). Reeves (1993) defines a heuristic as: "*a technique which seeks good (i.e. near-optimal) solutions at a reasonable computation cost without being able to guarantee either feasibility or optimality, or even in many cases to state how close to optimality a particular feasible solution is.*" Several common heuristic approaches are presented below:

5.2.1 Local Search - Hill-Climbing

Hill-climbing involves starting at a random point in the search space and then looking at the closest neighbour/ candidate solutions. If the value of a neighbour solution is better solution than the current solution, then it becomes the current node. If the neighbour is not a better solution, the current node remains the same. The process loops over and over until no better solution can be found i.e. a local maximum has been reached. It is possible to have more than one local optimum. In **Figure 6**, depending on the starting position, a local maximum may be reached and the algorithm would get stuck, but it may not be the global maximum i.e. the best solution. This is one of the major drawbacks of the hill-climbing algorithm.

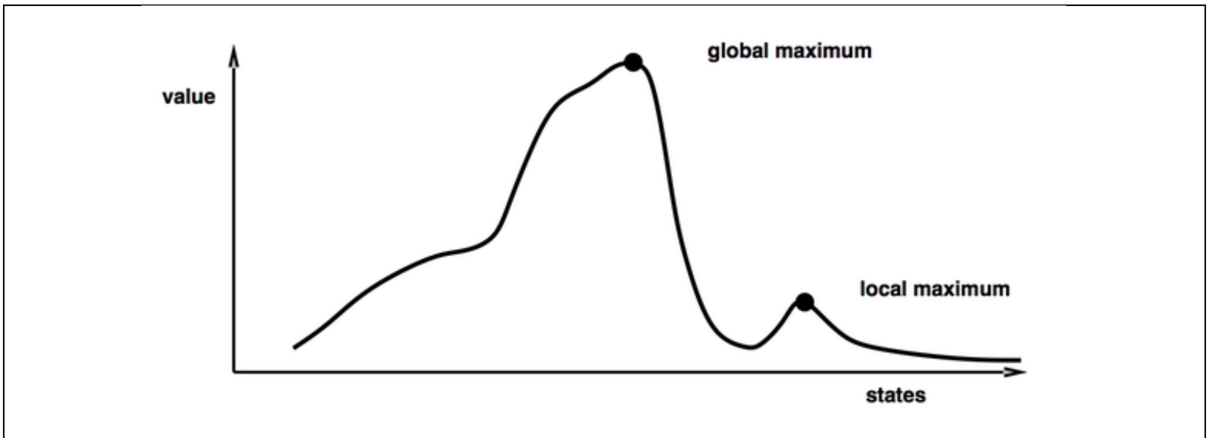


Figure 6: Demonstration of a local maximum being reached rather than the global maximum.

5.2.2 *Local Search – Hill-Climbing with Random Restart*

This algorithm fixes the problems associated with hill-climbing. The hill-climbing algorithm is run until a local maximum is found. It will then be run again starting from a different random position in the search space, which possibly enables a new local maximum to be found. This process is repeated and eventually the best optimum that was found will be returned.

5.2.3 *Local Search - Simulated Annealing*

This algorithm is based on the same idea as hill-climbing with restart but it is different in the sense that the neighbour chosen is not always an improvement to the current solution i.e. the algorithm may move downhill. This is because it looks further than just the neighbour solution – it takes a random walk in the search space to see if there are better solutions beyond local maximums in the hopes of discovering the global maximum i.e. searches getting stuck in at local maximums are prevented. Simulated annealing was inspired by metallurgy's annealing technique - a method used to reduce defects in a metal by using heating and controlled cooling down of the material. This search process is dependent on the variable 't' or 'temperature'. The algorithm begins with a high temperature, and slowly cools down to a low temperature. The higher the t, the more the search ignores local maximums while the lower the t, the more this algorithm behaves like the normal hill climbing algorithm.

5.2.4 *Local Search - Tabu Search*

Unlike the above local search algorithms, Tabu search makes use of memory which enables it to overcome local maximums. This heuristic is commonly used in the solving of mTSPs, TSPs and VRPs (Uldall, Taarnhøj and Vorts, 2008).

5.2.5 *Genetic Algorithm*

Genetic algorithms were inspired by processes observed in biological evolution and natural selection. In this algorithm, a solution to a problem can be viewed as an *individual* and a group of solutions becomes a *population*. Some sort of objective function, called the *fitness function*, is used to compare various individuals in a population. In an iterative process, new generations are formed from the best performing individuals in the previous generation as well as their offspring, while keeping the population size constant. New generations are created through selection, crossover and mutation of individuals from the previous population.

Potvin (1996) defines the steps to a simple "pure" genetic algorithm as follows:

TABLE 1: GENETIC ALGORITHM METHODOLOGY

Step 1. Create an initial population of P chromosomes (generation 0).
Step 2. Evaluate the *fitness* of each chromosome.
Step 3. Select P parents from the current population via proportional *selection*
Step 4. Choose at random a pair of parents for mating. Exchange bit strings with the one-point *crossover* to create two offspring.
Step 5. Process each offspring by the *mutation* operator, and insert the resulting offspring in the new population.
Step 6. Repeat steps 4 and 5 until all parents are selected and mated (P offspring are created).
Step 7. Replace the old population of chromosomes by the new one.
Step 8. Evaluate the fitness of each chromosome in the new population.
Step 9. Go back to step 3 if the number of generations is less than some upper bound. Otherwise, the final result is the best chromosome created during the search.

Some of the concepts mentioned above are described as follows:

Fitness: In order to evaluate how effective a solution is, and to compare effectiveness between solutions, a fitness function is required. In the case of a TSP, the shorter a solution's total travel distance, the more effective that solution is, and so the fitness of the solution would be a function of the total distance travelled.

Selection: The selection process of a genetic algorithm determines which individuals should mate and produce offspring for the next generation. The aim of selection is to choose individuals with high fitness values and to discard the poor performing ones. However, these bad individuals should also have a chance to be selected because they may lead to useful genetic material (Razali and Geraghty, 2011). Various selection methods exist:

- **Proportional roulette wheel selection** selects individuals in proportion to their fitness, so the higher an individual's fitness value, the more likely is it is to be selected (Luke, 2013).
- In **rank based roulette wheel selection**, solutions are ordered according to their fitness values and selection probabilities are calculated for each solution based on their ranks relative to the population rather than on their fitness value. This selection method is therefore not influenced by extremely high performing individuals, like the proportional roulette wheel selection is.

- **Tournament selection** is the most popular method due to its efficiency and simple implementation (Razali and Geraghty, 2011). k individuals are randomly chosen from the population and the individual with the best fitness value wins. The larger the value of k, the greater chance for loss of diversity in new generations (Blickle and Thiele, 1995). This is because poor performing individuals are unlikely to win large tournaments and therefore don't get a chance to be selected for mating. Smaller values of k give most individuals a chance to be selected and thus preserve genetic diversity. In a study performed by Razali and Geraghty (2011), it was found that tournament selection gave better results than proportional roulette wheel for all sizes of problems that were tested. It was also easier to implement than the rank based roulette wheel selection method.

Crossover: Crossover is the Genetic Algorithm's distinguishing feature (Luke, 2013). The best features of the parents chosen from the selection step are combined to create offspring. There are three well-known crossover methods, which are illustrated in **Figure 7**.

- **One-point crossover:** a random number is chosen between 0 and the length of the solution which is the cut point. The child will contain the values up until the cut point from parent 1, and the values after the cut point in parent 2.
- **Two-point crossover:** This method breaks up the solution at 2 points, creating a section which is swapped with the section created from the other parent.
- **Uniform crossover:** This method selects random positions in both the parents to swap their genes.

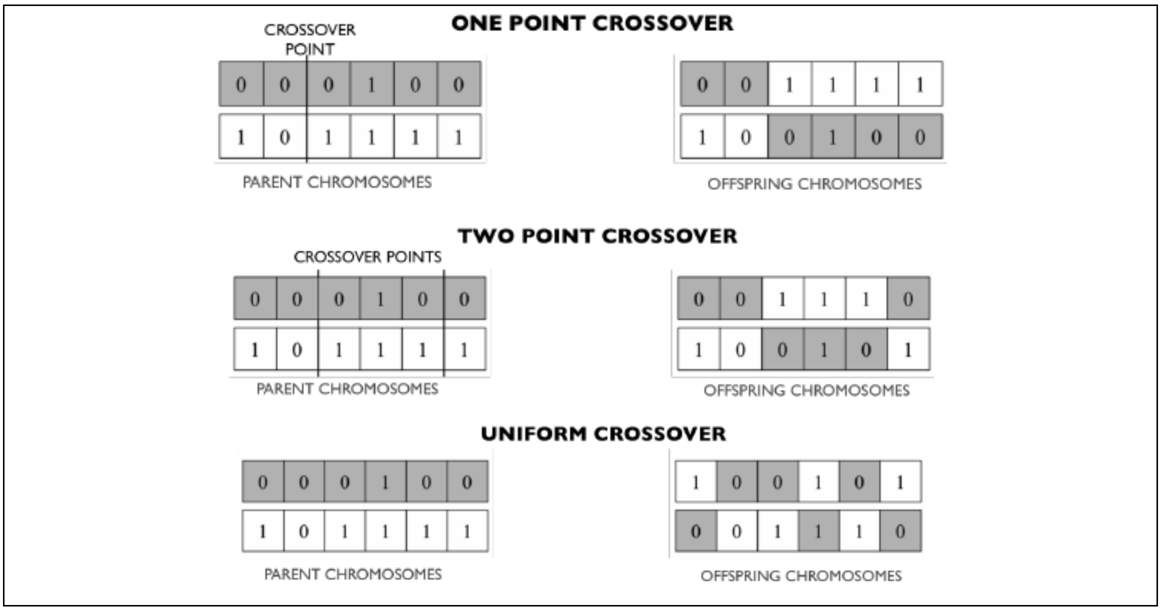


FIGURE 7: DEMONSTRATIONS OF COMMON CROSSOVER OPERATORS

When applying the pure genetic algorithm to a TSP, a problem is encountered when using any of the aforementioned crossover methods. After applying a crossover, some cities may appear more than once in the offspring created, making them infeasible solutions i.e. the child is not a permutation of its parents. Potvin (1996) describes the '**order 1 crossover'** method (as illustrated in **Figure 8**) which overcomes this issue. The aim of this technique is to preserve the ordering of the locations from both parents and it works in the following way:

1. Select a random part of the chromosome from parent 1 by randomly choosing 2 cut points
2. Drop this part down to child 1 and mark out these locations from parent 2
3. Add the remaining values from parent 2 to the child in the order in which they appear in parent 2, starting after the second cut point
4. If a second child is required, swap parent 1 and parent 2 and go back to step 1.

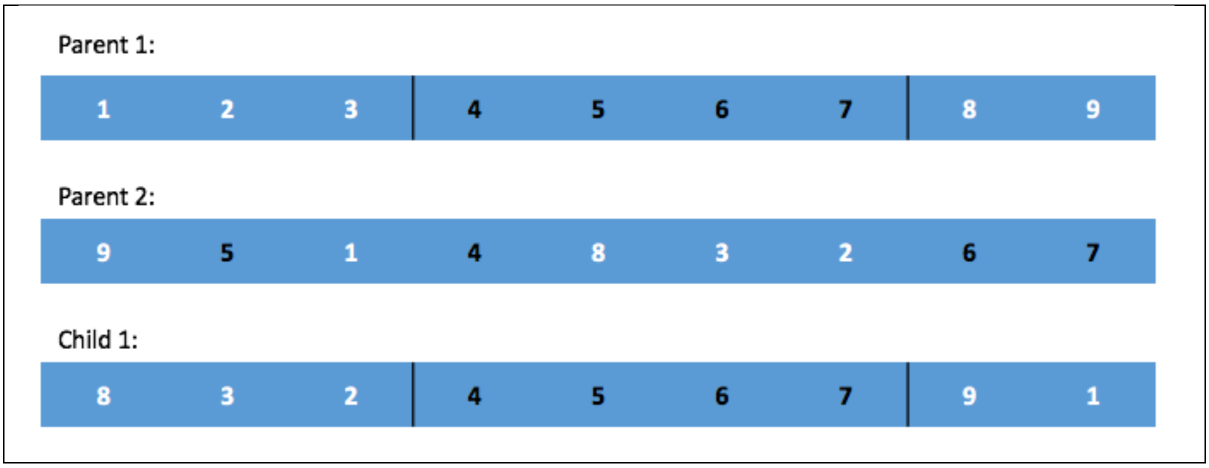


FIGURE 8: DEMONSTRATION OF ORDER 1 CROSSOVER

Mutation: In order to maintain genetic diversity through generations, a mutation of some individuals should occur with a small probability. Mutation prevents the algorithm from getting stuck at a local optima (Larranaga *et al.*, 1999). Examples of mutation methods include:

- **Swapping**: 2 locations in the chromosome are randomly selected and then swapped.
- **Scrambling**: Two cut points are randomly selected, and the locations within the two cut points are randomly permuted

Elitism: This concept is often used in genetic algorithms to ensure that the best performing individuals (the elites) appear in future generations. However this technique may cause premature convergence if not kept in check (Luke, 2013).

According to Pinedo (2012), using genetic algorithms is advantageous because they are easily coded and give good solutions, but their computation time can be longer than other heuristic approaches.

5.3 Conclusion

The problem with which Company XYZ is faced can be classified as a type of Travelling Salesman Problem. A variety of exact and heuristic methods exist to solve these problems, which are summarised and compared below:

Table 2: Comparison of methods for solving TSP

Method	Advantages	Disadvantages
Mathematical programming	Many formulations of TSP have been found	Almost impossible to find optimal solution for large problems Long running time
Dynamic programming	Existing Held-Karp method found in literature	Long running time Difficult to develop code High memory usage
Hill-climbing	Very simple algorithm	High chance of getting stuck at local optima
Hill-climbing with random restart	Local optima can be avoided	If random restart points are close, same local optimum will continually be reached
Simulated annealing	Easily coded Local optima can be avoided	Can be slow, especially if cost function is expensive to compute Simpler, faster methods exist Cannot tell if solution is optimal
Tabu search	Local optima can be avoided	Only works in discrete spaces Requires a lot of memory Cannot tell if solution is optimal
Genetic algorithm	Easily coded Known to have produced good solutions Local optima can be avoided A lot of literature found	Longer running times than other heuristic approaches Cannot tell if solution is optimal

Because it is such a large problem, heuristics appears to be the best way of finding a valid solution. Genetic Algorithm is an effective method which has commonly been used to solve TSPs. The genetic algorithm techniques found can be adapted for the specific use of the solving Company XYZ problem.

6. PROBLEM INVESTIGATION

Before Fourier-E's intervention, Company XYZ had no specialized programming or algorithmic methods to use in the scheduling of their employees. Each day a manually-created job sheet was given to an employee who would then have to visit and service each of the service requests in that day. If another job came up, it would simply be added on top of the pile. This meant that no thought went into the routing or the order in which jobs should be done and so there was a lot of travelling back and forth. This lead to employees working overtime which became costly for Company XYZ. As mentioned in the background of the problem, resource utilisation was also unbalanced. Some weeks, an employee would work much longer hours than in other weeks. These issues lead to the need for consulting Fourier-E to perform a resource planning project. From 2009, Fourier-E has worked on building models to improve maintenance task allocation, smooth out resource allocation and minimise travel times for Company XYZ. Their final linear-programming model was used to schedule employees over an 80-week period for all the regions in which Company XYZ worked. Its logic worked in the following way:

1. A resource is selected
2. A particular week is selected
3. The site with the most equipment is selected to be serviced first in this week
4. Thereafter, the closest site with the most equipment is selected
5. The time available is checked to see if the resource can perform the service
6. The equipment to be serviced is selected, ordered by priority, and time available
7. The maintenance plan of the current equipment is updated

The output of the model is an employee job sheet which included the following information:

- Resource number
- Week number
- List of equipment to be serviced
- Total travel time
- Total task time
- Total time

Resource Number	Week	Day	Site Previous	Site Current	Equipment	Travel Time	Task Time	Total Time
3755927	311	1	6000-26239	6000-07951	30003578, 30007634, 30008426, 30003577, 30007226, 30007227, 30007228	4	265	269.0
		1	6000-07951	6000-06451	10010734	12.2	15	296.2
		2	6000-26239	6000-06451	30003279, 30003280, 30003288, 30003277, 30003278, 30003274, 30003273, 30003281, 30003282, 30003283	16.2	280	592.4
		3	6000-26239	6000-06451	30003284, 30003265, 30003266, 30003267, 30003268, 30003285	16.2	245	853.6
		4	6000-26239	6000-06451	10010732, 10010733, 30003275, 30003276, 30003286	16.2	255	1124.8
		5	6000-26239	6000-06451	30003287, 30003269, 30003270	8.1	255	1387.9
						5%	95%	1387.9

FIGURE 9: AN EXAMPLE OF AN EMPLOYEE JOB SHEET CREATED BY FOURIER'S MODEL

A graph has been created to show the costs associated with the use of the modelled schedule over the 80-week period. This chart illustrates the imbalance of workload over the 80 weeks. There are still periods with high utilisation of employees and periods where there is very little utilisation.

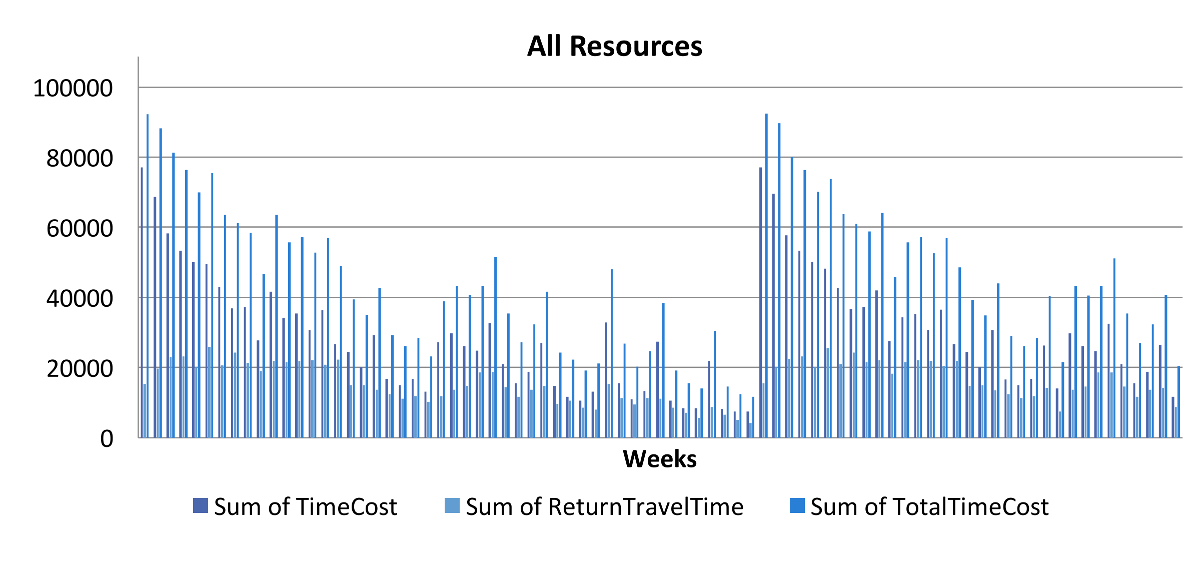


FIGURE 10: CHART SHOWING COST OF SCHEDULE OVER 80 WEEKS

The charts below compare the schedules of two employees. The red line represents the employees' available time to work in a week. The charts clearly show that employees are still not being fully utilized, and their schedules have not balanced their workloads over the 80-week period. Also Resource 3755927 has a much busier schedule than Resource 1212551, even though he/ she has less time available to work.

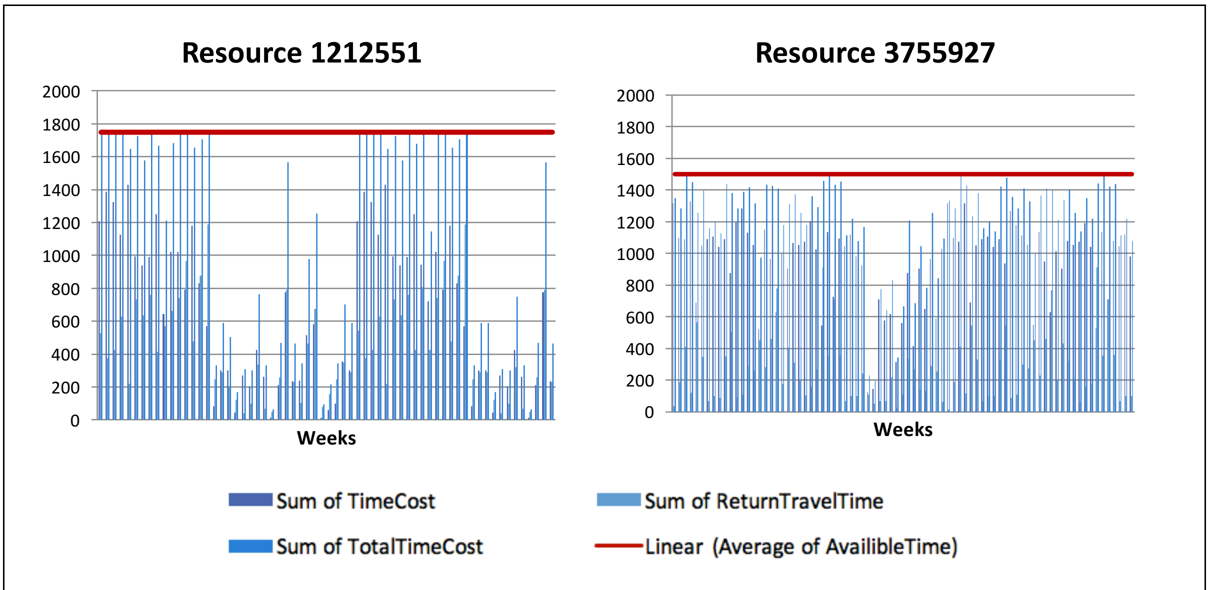


FIGURE 11: CHARTS COMPARING GENERATED SCHEDULES OF 2 EMPLOYEES

The model works for Company XYZ and they are currently using it to automate their scheduling process but there is still there is much room for improvement. Fourier's model can therefore be used as a base on which to improve in order to solve the problems of Company XYZ. The following considerations have been made regarding the current and potential model:

- Linear programming was used to create Fourier's model. Because of the large number of variables and constraints, this model took days to solve, which makes it impractical to use on a daily or weekly basis. Literature reveals that metaheuristics is a useful tool in solving large, complex problems and so this should be implemented in the new model
- Fourier's model worked in terms of weeks. This allows for the freedom of refining the weekly schedule if emergencies need to be accommodated. However, daily scheduling provides more accurate travel times and so it should be incorporated into the new model
- The model created a static plan for 80 weeks. In this long time frame, a lot can happen and change which makes this long term plan impractical. If changes should occur, or emergency services need to scheduled, then the model should be able to reorganise existing schedules.
- The robustness of the schedule needs to be considered. Is it better to have a rough plan which guides employees but they can make changes if they need to, or to have a very detailed plan which an employee has to stick to?
- Emergency breakdowns or issues can happen at any point in time, which the model needs to take into account. Is it better to have for example 7 resources with 100% utilisation and a separate team to deal with emergencies, or to have 10 resources with 75% utilisation so that they are available when emergencies occur?

All these points will be considered in the development of the final model, which is presented in the following section.

7. SOLUTION DEVELOPMENT

This section describes the chosen algorithm and its development in detail.

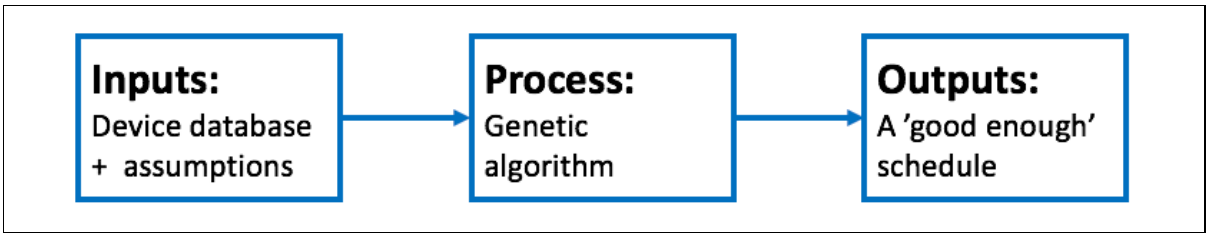


FIGURE 12: THE BASIC CONCEPTUAL MODEL PROCESS WHICH WILL BE FOLLOWED

7.1 INPUT

The Database

The device database is an excel spreadsheet containing information about each device that Company XYZ is required to service. This database will be converted into a csv file to be used as an input to the algorithm. Important attributes of each equipment which the algorithm will use in its calculations include the device's identification number, service time (minutes), as well as its longitude and latitude. A preview of this database is shown in the figure below:

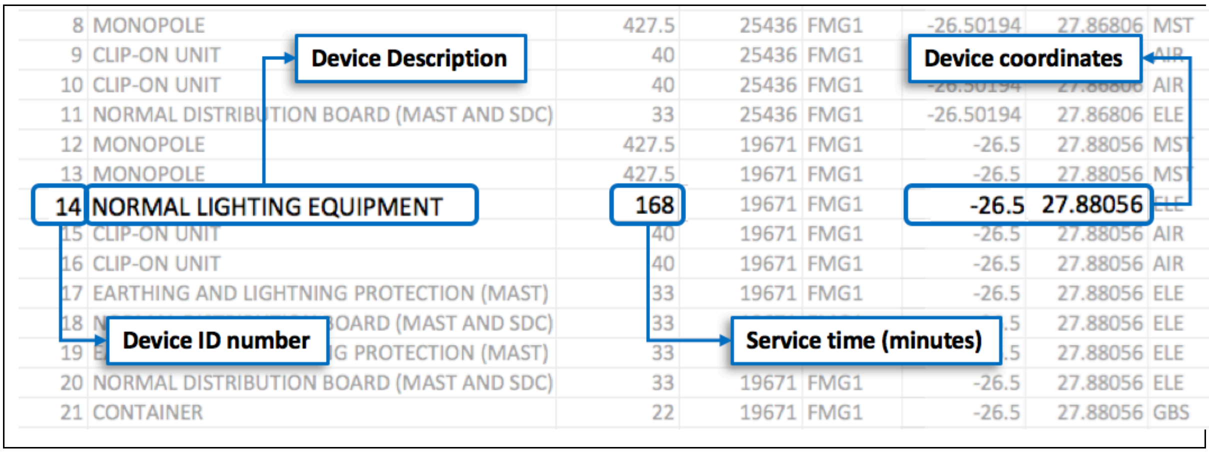


FIGURE 13: PREVIEW OF IMPORTANT ASPECTS OF DATABASE

There are 20379 electrical devices in the database that need to be serviced. In the Travelling Salesman Problem, the number of possible routes is a factorial of the number of cities to visit. If each electrical device were to be treated as a city, there would be 20378! = 8.02 E+78968 routes to consider.

Many of the devices are located in the same buildings, and so to simplify the problem slightly, the electrical devices in the database will be ordered according to their coordinates. Devices in the same locations will then be grouped together, and a new input csv file will be created such as the one in **Figure 14** using a simple Python program. This will ensure that when creating random individuals for the population, devices in the same location will not be separated, and the algorithm will then optimize distances between locations rather than distances between the devices themselves.

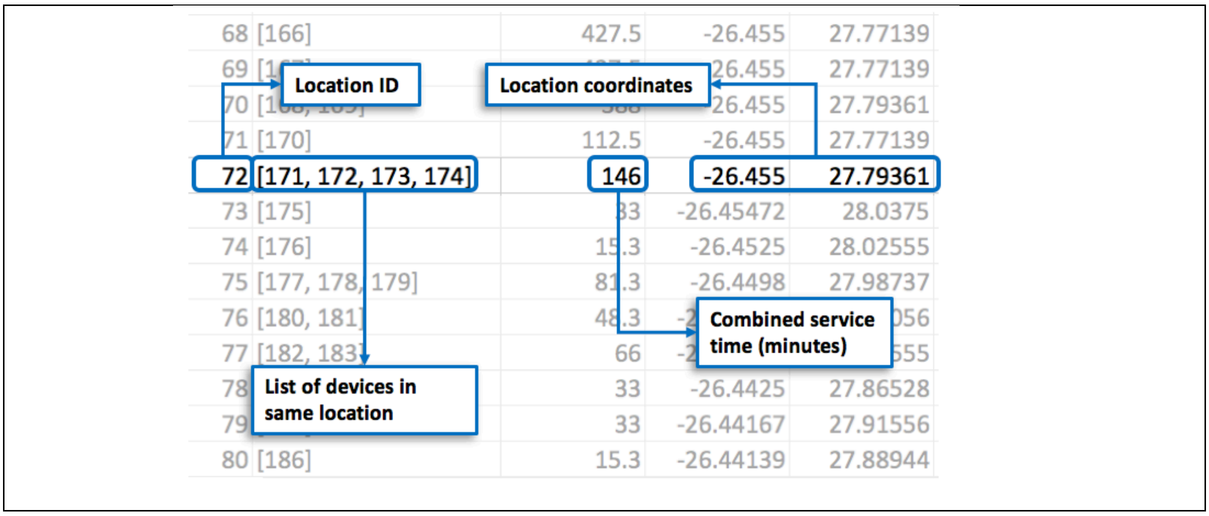


FIGURE 14: DATABASE WHICH GROUPS DEVICES COMMON TO A SPECIFIC LOCATION

After applying the grouping program there are 7778 locations, labelled 1 to 7778, which means there are now only 7778! = 3.74 E+26887 routes to consider.

Assumptions

The following assumptions will be applicable in the building of the genetic algorithm:

- 36 full-time technicians are currently employed by Company XYZ
- Technicians work Monday through Friday
- Technicians should work roughly 8 hours per day
- All technicians have the required skills to service any electrical device
- Technicians drive at an average speed of 70 km per hour (vehicles make use of main roads where possible)
- Technician drives back to depot after their last job in a day
- Servicing of a device has to be completed the day it was started
- Servicing of any device requires only one technician
- Straight line distances are used between two locations

7.2 Process

The genetic algorithm will be built in *Python 3.6* using the methodology below, adapted from **Table 1** to apply to a Multiple Travelling Salesman Problem. Each step of the Genetic Algorithm will then be described in more detail. Note that the complete python coding for the algorithm can be found in **Appendix B.**

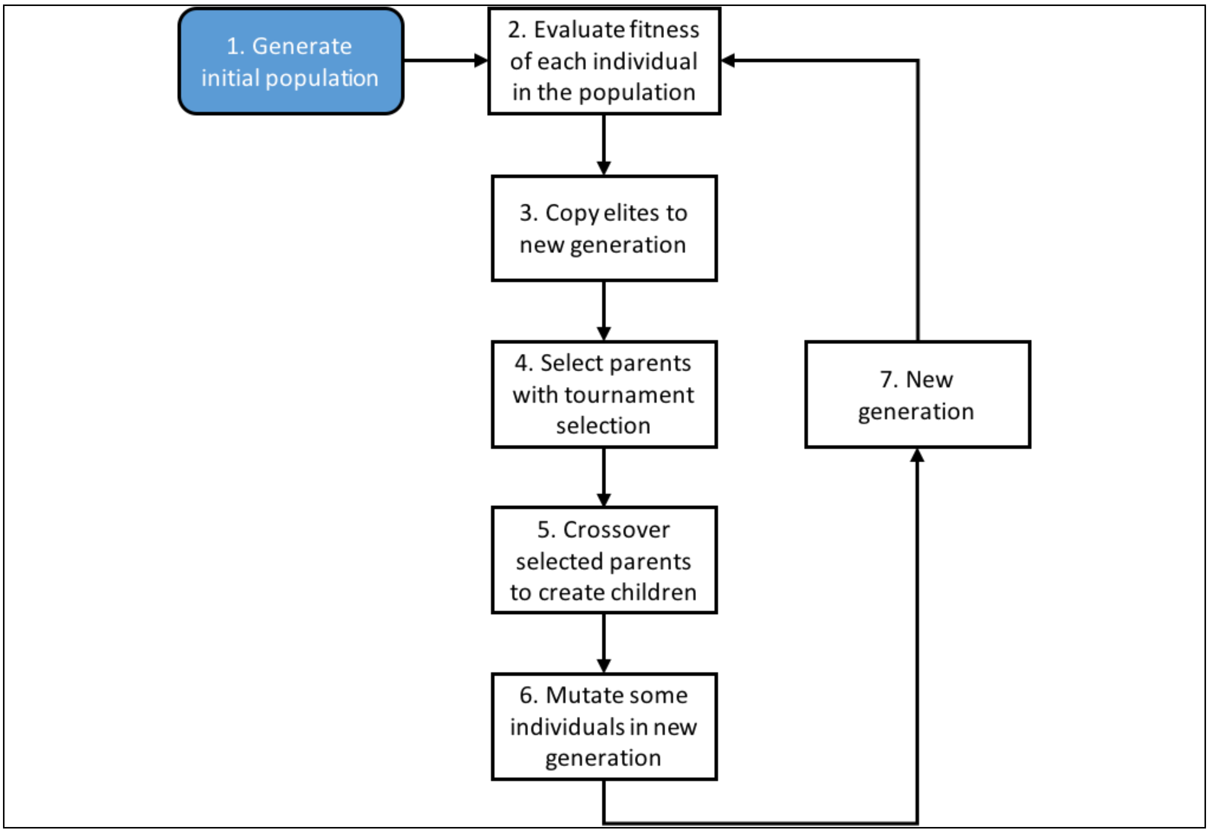


FIGURE 15: GENETIC ALGORITHM METHODOLOGY

1. Generate initial population

Firstly, the solution to the problem (chromosome) will be represented as a list of location IDs, in which each location represents the devices it holds. Each location is listed exactly once. This satisfies the first objective of the model: to satisfy all customer demand. i.e. service each device once.

1	2	3	…	7776	7777	7778

FIGURE 16: REPRESENTATION OF AN INDIVIDUAL SOLUTION

Now, **Algorithm 1** generates a population by taking this initial solution as an input and creating 100 random permutations of it:

Algorithm 1: Population

```
Input: Initial solution (ind)
Output: Population consisting of s permutations of initial solution

def population(ind, s):
    return [random.sample(ind, len(ind)) for i in range(s)]
```

2. Evaluate fitness of each individual of the population

In order to compare these different solutions and determine which are better than others, a fitness function is required. Going back to the problem statement, the main objectives of the model are to 1) reduce travel times and 2) even out workload distribution. The fitness function therefore needs to be comprised of two elements: 1) total distance of the schedule and 2) the total deviation from 8 hours of working time per resource per day. This fitness function will take an individual solution as an input, and return that solution's total distance and deviation. The flow diagram below demonstrates the logic behind the fitness function:

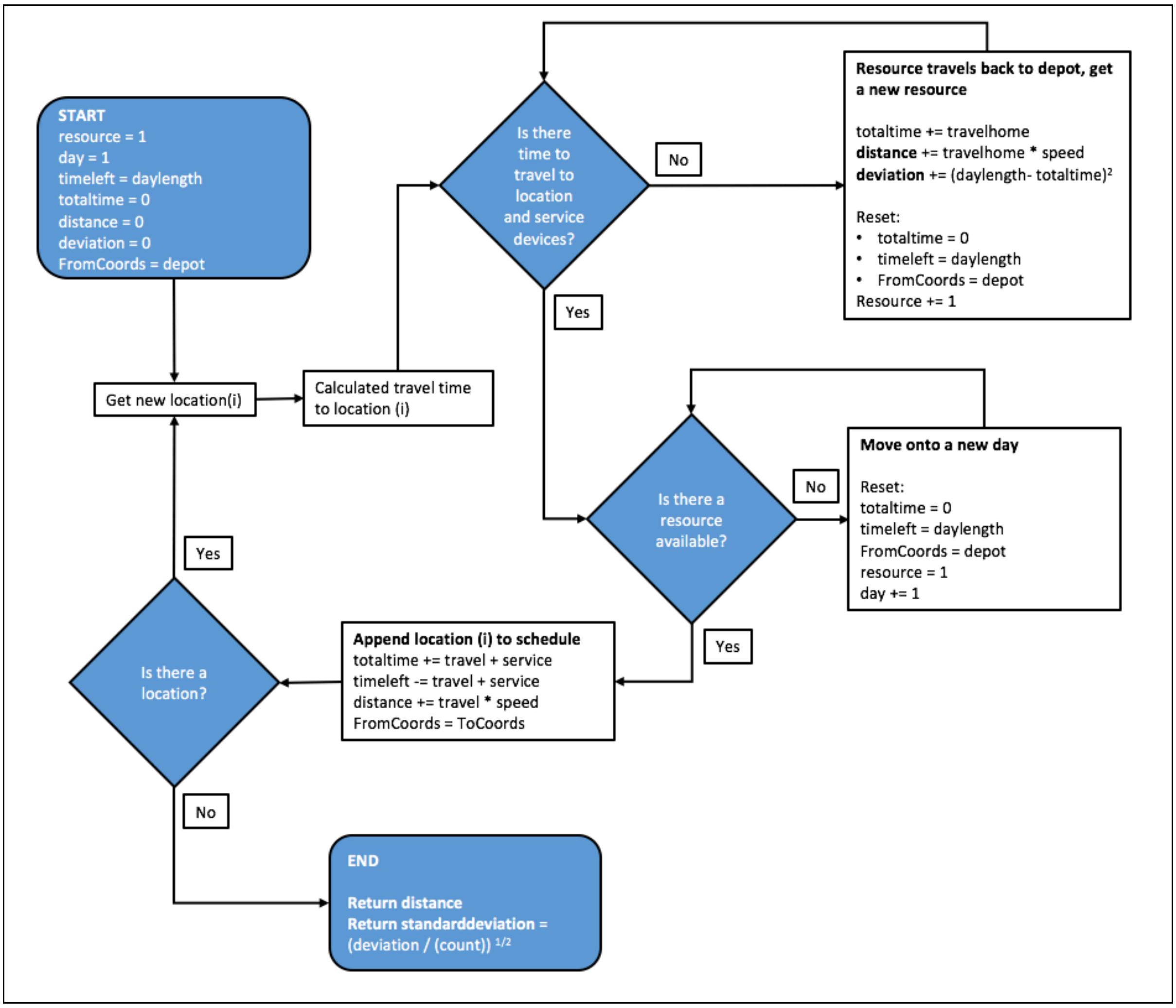


FIGURE 17: FLOW CHART DEMONSTRATING THE LOGIC BEHIND THE FITNESS FUNCTION

Firstly, the travel time to the first location listed in the solution from the depot is calculated. The following algorithm will be used to calculate the straight line distances between 2 locations, and then be multiplied by the speed of the vehicle to find the travel time:

```
Algorithm 2: Distance

Input: Origin coordinates and destination coordinates
Output: Straight line distance in km between 2 input locations

def distance(origin, destination):
    lat1, lon1 = origin
    lat2, lon2 = destination
    radius = 6371
    dlat = radians(lat2 − lat1)
    dlon = radians(lon2 − lon1)
    a = sin(dlat / 2) ∗ sin(dlat / 2) + cos(radians(lat1)) ∗
    cos(radians(lat2)) ∗ sin(dlon / 2) ∗ sin(dlon / 2)
    c = 2 ∗ atan2(sqrt(a), sqrt(1 − a))
    d = radius ∗ c
    return d
```

If there is enough time left in the day for the resource to drive there and perform the servicing, then the resource will travel to that location. The travel time and servicing times are then added to the total working hours, and the total distance for the schedule is updated. When there is not enough time left in the day for a resource to drive to a location, he will drive back to the depot, and then a new resource will be selected to travel to that location. Once all 36 resources are used up in a day, a new day begins with resource 1.

Note that the 'is there time to travel to location and service devices' decision block does not check if there is also time to drive back to the depot once servicing is complete. This means that the drive back to the depot may cause an employee's day to run over 480 minutes. When a resource drives back to the depot, his total time, service time and travel time for that day are printed. This is when the deviation of the schedule is calculated:

$Deviation = \sqrt{\frac{\sum(480 - total\ time)^2}{Count}}$ where 'count' is the total number of schedules created.

3. Copy elites to new generation

This process will be described in **step 7**

4. Select parents with tournament selection

Tournament selection will be used as the method of selecting parents for breeding. Randomly select k individuals from the population and return the best performing individual from this group.

```
Algorithm 3: Selection (Tournament selection)

Input: Population (P) and tournament size (k)
Output: Winning individual of the tournament

def selection(P, k):
    best = null
    for i in range(1, k):
        ind = random individual in P
        if (best == null) or fitness(ind) > fitness(best):
            best = ind
    return best
```

5. Crossover selected parents to create children

Order one cross over was selected as the method of performing a crossover to prevent invalid solution from being created.

```
Algorithm 4: Crossover (Order one)

Input: Two individual parents, p1 and p2
Output: Child containing characteristics from both parents, with no location
repetitions

def crossover(p1, p2):

    r1 = random number between 0 and length of length of parents
    r2 = random number between 0 and length of length of parents
    child = list with length equal to length of p1
    Copy elements between r1 and r2 from p1 to child, in same positions as p1
    y = List holding elements of p1 which are not in child yet
    Order the elements in y according to their order in p2
    Copy remaining elements into child according to their order in p2
    starting after r2

    return child
```

6. Mutate some individuals in new generation

```
Algorithm 5: Mutation (Swapping)

Input: An individual (ind)
Output: Individual with 2 locations swapped around

def mutation(ind):
    x = random integer between 0 and length of individual
    y = random integer between 0 and length of individual
    ind[x], ind[y] = ind[y], ind[x]

    return ind
```

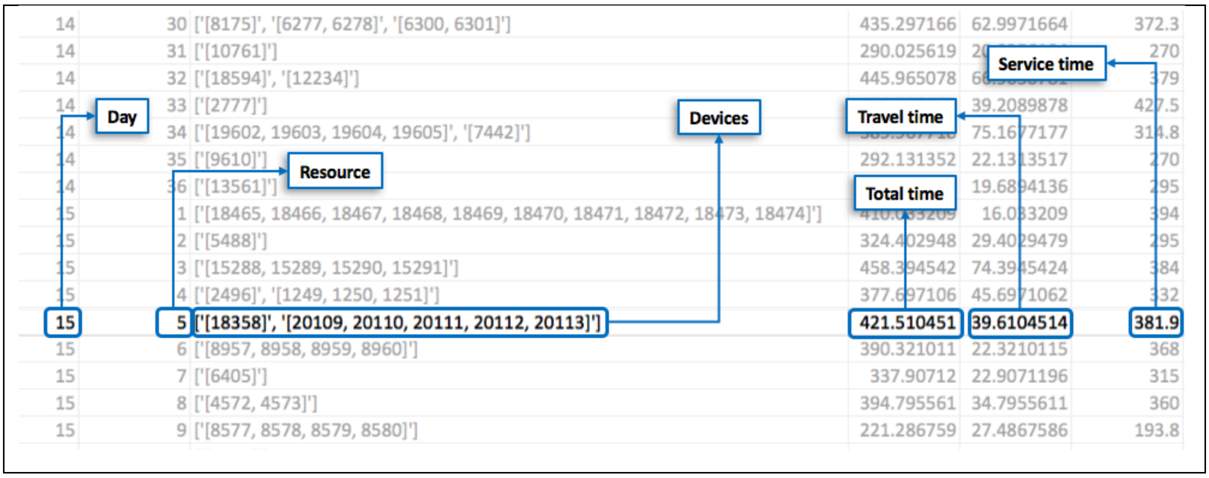
7. New generation

An 'evolve' function is created by combining selection, crossover and mutation. The individuals of the population will be sorted according to their fitness values. The top 20% of the population will be copied to the next generation as they are [Step 3: Copy elites to new generation]. Next, random individuals will be chosen from the current generation to produce 2 children. The elites may also be selected for breeding. Once crossover occurs, there may be a chance that the children are mutated. This process will loop through until the next generation is large enough.

```
Algorithm 6: Evolution

Input: Population (P) sorted in order of fitness, mutation probability (m),
elitist percentage (r)
Output: New population

def evolve(P, m = 0.01, r = 0.2):

    retain_length = length of P * r
    parents = P[:retain_length]
    parents_length = length of parents
    desired_length = length of P - parents_length
    children = []

    while length of children < desired_length:

        p1 = selection(p, 3)
        p2 = selection(p, 3)

        if p1 is not the same as p2:
            child1 = crossover(p1, p2)
            child2 = crossover(p2, p1)
            append child1 to parents
            append child2 to parents

    for i in parents:
        if m > randomly generated number between 0 and 1:
            mutate(i)

    sort individuals in parents according to fitness

    return parents
```

7.3 Output

The evolution function will be run 100 times to produce 100 generations. The best performing solution of the 100th generation will be exported as a csv file and analysed. The solution indicates which resources will be performing services on which devices, on which day, how long their day will be and what proportions of it will be spent travelling and actually performing the service. A preview of a solution can be seen in the following figure:



14	30	['[8175]', '[6277, 6278]', '[6300, 6301]']	435.297166	62.9971664	372.3
14	31	['[10761]']	290.025619	[illegible]	270
14	32	['[18594]', '[12234]']	445.965078	[illegible]	379
14	33	['[2777]']	[illegible]	39.2089878	427.5
14	34	['[19602, 19603, 19604, 19605]', '[7442]']	[illegible]	75.1677177	314.8
14	35	['[9610]']	292.131352	22.1313517	270
14	36	['[13561]']	[illegible]	19.6894136	295
15	1	['[18465, 18466, 18467, 18468, 18469, 18470, 18471, 18472, 18473, 18474]']	[illegible]	16.0[illegible]3209	394
15	2	['[5488]']	324.402948	29.4029479	295
15	3	['[15288, 15289, 15290, 15291]']	458.394542	74.3945424	384
15	4	['[2496]', '[1249, 1250, 1251]']	377.697106	45.6971062	332
15	5	['[18358]', '[20109, 20110, 20111, 20112, 20113]']	421.510451	39.6104514	381.9
15	6	['[8957, 8958, 8959, 8960]']	390.321011	22.3210115	368
15	7	['[6405]']	337.90712	22.9071196	315
15	8	['[4572, 4573]']	394.795561	34.7955611	360
15	9	['[8577, 8578, 8579, 8580]']	221.286759	27.4867586	193.8

FIGURE 18: PREVIEW OF A SOLUTION IN EXCEL FORMAT

The output will also show how many days will be required to complete servicing of all electrical devices as well as the schedule's total distance travelled and total deviation.

8. Solution and Solution Validation

This section presents the final solution as well as how Company XYZ should go about the process of verifying and validating the final model.

Graphs have been created to show how both the total distance and total deviation of the solutions decreased with every new generation created. Both the best and worst fitness values of each generation were plotted, as can be seen in the graphs below:

FIGURE 19: GRAPHS DEPICTING THE IMPROVEMENT OF SOLUTIONS WITH EVERY NEW GENERATION

To show how the genetic algorithm performed, the best solution of the final generation is compared with the best solution of the initial generation in **Table 3** on the following page:

TABLE 3: COMPARISON OF BEST INITIAL SOLUTION AND BEST FINAL SOLUTIONS

	Initial Solution	Final Solution
Total overtime	91.206 hours	91.296 hours
Total idle time	13 872.654 hours	13 669.592 hours
Total distance	316 843.8 km	312 328.59 km
Total deviation	142.86 minutes	140.45 minutes
Days to complete	189 days, 36 resources	189 days, 7 resources

Overtime hours increased only very slightly but idle time has decreased by 203 hours. This has allowed the servicing of all the devices to take place in 189 days, using only 7 resources on the 189th day rather than the initial 36. The graphs below show a randomly selected employees' working hours over the period of the schedules. The average length of a working day has increased from the initial solution to the final solution which means that the days for this employee are more full i.e. there is less idle time.

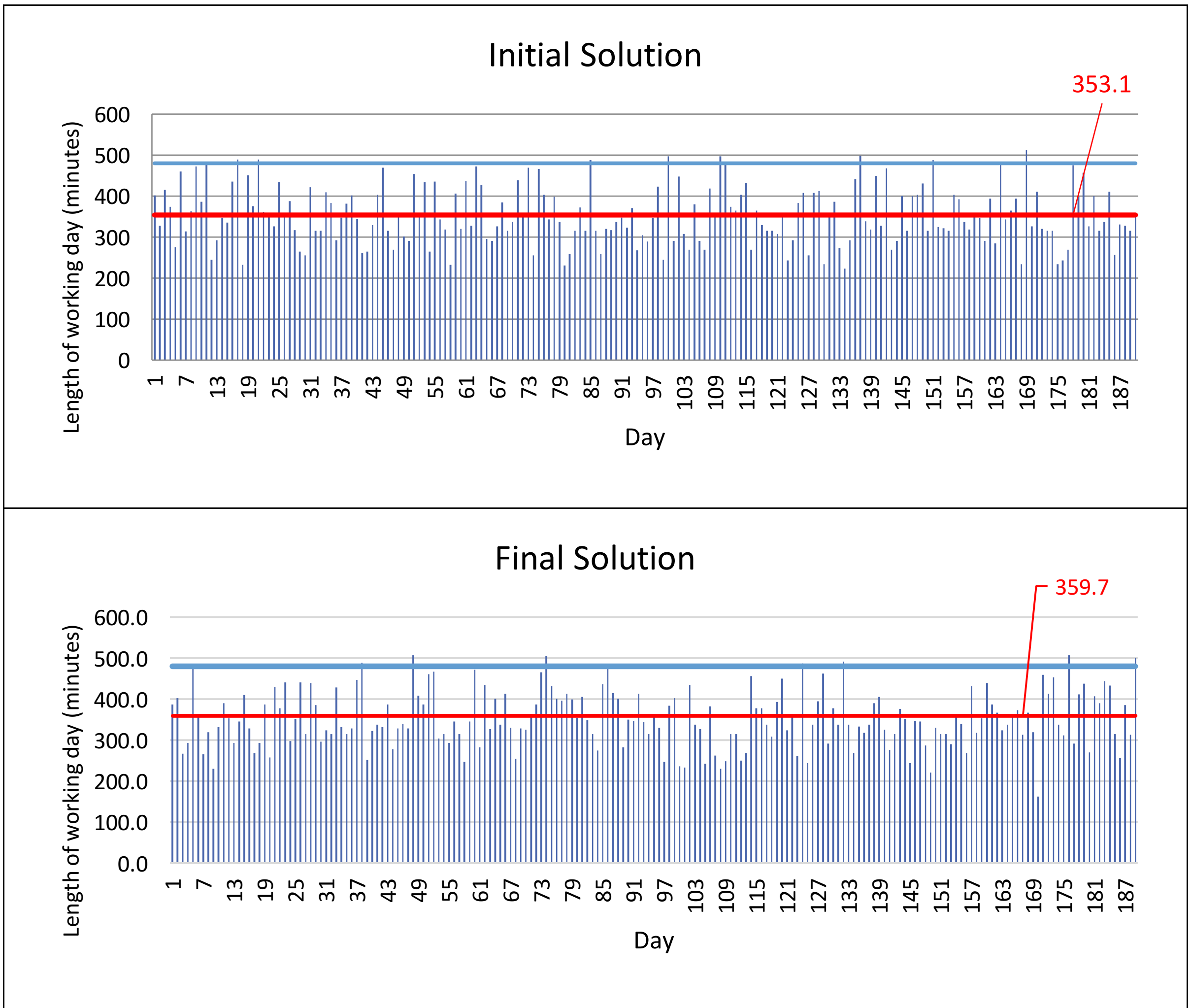


FIGURE 20: COMPARISON OF WORKING DAYS OF INITIAL AND FINAL SOLUTIONS FOR A PARTICULAR EMPLOYEE

There has been a reduction in the deviations from an 8-hour working day, but there is still visibly a lot of idle time. This can be used to the company's advantage - any employee with free time on their schedule can be put "on call" for any emergencies that may occur.

Has the genetic algorithm addressed Company XYZ's issues?

The way the algorithm is built does not allow a resource to travel to location if there won't be enough time to service the devices, and so by using this algorithm, Company XYZ can automatically cut down on their overtime costs.

The algorithm produces daily schedules for each employee rather than weekly. This means that the travelling times are more accurate and the schedule is more specific.

The genetic algorithm produces a solution within hours while the linear programming method that was being used took days. This means that the company can use the algorithm for more dynamic planning because it is more efficient and easier to keep up with continuous changes. If any device services pop up that were not initially planned for, the algorithm can be rerun to accommodate these devices in a more optimal way. Company XYZ can run the model it at the beginning of each week to create schedules, instead of following a static 80-week plan.

Solution Validation

The schedules produced by the genetic algorithm should be validated before the company starts implementing them. The company can put together a testing team for a week or two which follow the schedule as it is, and then report back any issues they may have had. Parameters such as travelling speed, and time in the day may have to be adjusted if employees find that they cannot manage with the given schedule. A benefit of using this algorithm is that it only requires Python which is a free, open source programming software as well as a person with a background of coding. The code can on a continuous basis be easily be adapted and modified.

9. PROPOSED IMPLEMENTATION

The csv output file that was produced by the algorithm can be used in a number of ways. Firstly, the excel table can be filtered to show the schedule of each employee. Resource 17 for example, has the following schedule for two weeks:

TABLE 4: SAMPLE SCHEDULE FOR AN INDIVIDUAL EMPLOYEE

Day	Devices	Total Time	Travel time	Service time
75	11296, 11297, 11298, 11299, 11300, 11301, 11302, 11303, 11304, 11305, 11306, 11307, 11308, 11309, 11310, 11311, 11312, 11313, 11314, 11315, 11316, 11317, 11318, 11319, 11320, 11321, 11322, 11323, 11324	431.9	20.0	411.9
76	10954, 10955, 10956, 10957	400.0	20.0	380
77	6121	395.5	80.5	315
78	3480, 3481, 3482, 3483	412.7	32.7	380
79	7514, 7515, 7516, 7517	398.4	18.4	380
80	3306	354.8	39.8	315
81	19914, 19915, 19916, 19917	405.7	25.7	380
82	5329	348.7	53.7	295
83	10564	315.0	20.0	295
84	20183	273.9	63.9	210

The excel table can also be filtered to show each day. Managers of Company XZY can use this to see which of their employees will servicing which devices. The table below is a sample for some of the employees on day 1.

TABLE 5: SAMPLE SCHEDULE FOR A SPECIFIC DAY

Resource	Devices	Total time	Travel time	Service time
13	['[12086, 12087, 12088, 12089]']	401.9	21.9	380
14	['[2428, 2429]']	424.1	64.1	360
15	['[4276]']	268.7	58.7	210
16	['[16432, 16433]']	422.8	32.8	390
17	['[13703, 13704, 13705, 13706]']	400.7	19.7	381
18	['[3946, 3947, 3948, 3949]']	438.4	58.4	380
19	['[2424]', '[17003]']	391.7	69.2	322.5
20	['[18285]']	331.5	16.5	315
21	['[19976, 19977]']	410.6	22.6	388

As mentioned in **Section 8**, the algorithm must first be put to the test to ensure that it is benefitting the company and that the schedule it has produced is verified and validated. Once the scheduling method has been implemented at the Gauteng branch and used for a period of time, users of this system can start making suggestions and improvements, and get rid of bugs. Thereafter, a finalised algorithm can be adopted by other regions in the country.

10. CONCLUSION

Company XYZ is having trouble scheduling their staff and determining which routes its technicians should be taking in order to satisfy customer demand, while incurring minimal costs. This problem was addressed by conducting a literature review to determine what type of problem was at hand and what methods have been developed to solve similar problems. The problem can be classified as a type of Travelling Salesman problem. It was found that the Genetic Algorithm proves to be a very useful efficient method of solving these problems. Using *Python 3.6*, a genetic algorithm was built and run 100 times to create 100 generations. With each new generation, a better scheduling solution was found i.e. schedules which reduced travel distances and balanced technician workloads.

And so, going back to the problem statement, has the developed solution met its objectives? *"Company XYZ requires a system which plans staffing in such a way as to balance the workloads of their employees, reduce costs and meet customer demand."* A genetic algorithm has been created [*a system which plans staffing*] which iteratively reduces solutions' deviation from an 8 hours' day of work [*balance workloads*] and total travelling distance, thereby saving on petrol, vehicle maintenance costs and overtime [*reduce costs*]. The algorithm ensures that every electrical device will get serviced [*meet customer demand*].

Before any schedule is implemented Company XYZ should go through a validation phase to ensure that assumptions made in the model are suitable. Using the genetic algorithm to create schedules could be advantageous for Company XYZ because it reduces overtime, plans daily, and finds solutions quickly.

11. BIBLIOGRAPHY

Blickle, T. and Thiele, L. (1995) 'A Comparison of Selection Schemes used in Genetic Algorithms', *Evolutionary Computation*, 2(11), pp. 311–347. doi: 10.1162/evco.1996.4.4.361.

Dorne, R., 2008. *Service Chain Management: Technology Innovation for the Service Business.* s.l.:s.n.

Haksever, C. *et al.* (2000) *Vehicle Routing and Scheduling*, *Service Management and Operation*.

Kokemuller, N., n.d. *Advantages & Disadvantages of a Part-Time Employees.* [Online] Available at: http://smallbusiness.chron.com/advantages-disadvantages-parttime-employees-21870.html
[Accessed 25 March 2017].

Laporte, G. (1992) 'The traveling salesman problem: An overview of exact and approximate algorithms', *European Journal of Operational Research*, 59(2), pp. 231–247. doi: 10.1016/0377-2217(92)90138-Y.

Larranaga, P. *et al.* (1999) 'Genetic Algorithms for the Travelling Salesman Problem: A Review of Representations and Operators', *Artificial Intelligence Review*, 13(2), pp. 129–170. doi: 10.1023/A:1006529012972.

Lew, A. and Mauch, H. (2007) '1 Introduction to Dynamic Programming', *Studies in Computational Intelligence*, 38, pp. 3–43. Available at: www.springerlink.com.

Linton, I., 2012. *Taking Technology to the Market: A Guide to the Critical Success Factors in Marketing Technology.* s.l.:Gower Publishing.

Luke, S. (2013) *Essentials of Metaheuristics*. 2nd edn, *Optimization*. 2nd edn. doi: 10.1007/s10710-

O'Brien, J., 2017. *What is scheduled maintenance critical percent?.* [Online] Available at: https://www.fiixsoftware.com/blog/scheduled-maintenance-critical-percent/
[Accessed 26 March 2017].

Pinedo, M. L. (2012) *Scheduling: Theory, Algorithms, and Systems*. Fourth, *Springer*. Fourth. London. doi: 10.1007/s13398-014-0173-7.2.

Potvin, J.-Y. (1996) 'Genetic Algorithms', *Annals of Operations Research*, 63, pp. 339–370. doi: 10.1016/B978-0-12-804494-0.00010-3.

Razali, M. and Geraghty, J. (2011) 'Genetic algorithm performance with different selection strategies in solving TSP', *World Congr. Eng.*, II(978-988-19251-4–5), pp. 4–9. Available at: http://umpir.ump.edu.my/2609/.

Reeves, C. R., 1993. *Modern Heuristics Techniques for Combinatorial Problems.* London: Blackwell Scientifici Publicaitons.

Uldall, C. P., Taarnhøj, E. S. and Vorts, S. (2008) *Technician Routing and Scheduling*. Denmark. Available at: http://orbit.dtu.dk/getResource?recordId=211562&objectId=1&versionId=1.

Winston, W. (2003) 'Introduction to Linear Programming', in *Operations Research: Applications and Algorithms*. 4th edn, pp. 49–125.

12. APPENDICES

APPENDIX A: SIGNED 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 *UPSpace* and 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 information is 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. Ensures that any sensitive, confidential information or intellectual property of the company is not disclosed in the Final Project Report.

Project Sponsor Details:

Company:	Fourier-E (Part of Fourier Approach)
Project Description:	A national IT company is using full time and part time resources to perform routine maintenance on electronic devices at their customers' premises. Each premises has multiple devices which requires a specific skill to perform the maintenance. A scheduling algorithm is to be developed to determine the maintenance plan for a year. In addition, the key cost drivers are to be identified, resource utilisation measured and company's maintenance costs reduced.
Student Name:	Kirsten Young
Student number:	14062471
Student Signature:	KYoung
Sponsor Name:	Rikus Kellerman
Designation:	Industrial Engineer.
E-mail:	Kellermanr@fourier.co.za
Tel No:	+27(0) 12 667 3232
Cell No:	+27 (0) 82 420 4957
Fax No:	F: +27(0) 12 667 3244
Sponsor Signature:	

APPENDIX B: COMPLETE PYTHON CODING

```
import math
import csv
import random
import null

daylength = 8 * 60
employed = 36
depot = -26.117920, 28.135320
speed = 70 / 60

with open("/Users/kirstenyoung/Google Drive/University/4th
Year/BPJ/Data/CSVs/GroupedData.csv") as csvfile:
    ImportEquip = list(csv.reader(csvfile, delimiter=';'))
    eqtdetails = {}
    initialsolution = []


# Initial Solution
def initialise():
    counter = 0
    for e in ImportEquip:
        counter += 1
        initialsolution.append(counter)
        eqtdetails[counter] = e
    return initialsolution


def devices(location):
    return eqtdetails[location][1]


# Calculate distance between 2 equipment locations
def d(origin, destination):
    lat1, lon1 = origin
    lat2, lon2 = destination
    radius = 6371
    dlat = math.radians(lat2 - lat1)
    dlon = math.radians(lon2 - lon1)
    a = math.sin(dlat / 2) * math.sin(dlat / 2) + math.cos(math.radians(lat1)) *
math.cos(
        math.radians(lat2)) * math.sin(dlon / 2) * math.sin(
        dlon / 2)
    c = 2 * math.atan2(math.sqrt(a), math.sqrt(1 - a))
    d = radius * c
    return d


# Return the coordinates of a device
def coords(device):
    return float(eqtdetails[device][3]), float(eqtdetails[device][4])


# Return the service time of a device
def serv(device):
    return float(eqtdetails[device][2])


# Rotate elements ina list
def rotate(lst, x):
    lst[:] = lst[x + 1:] + lst[:x + 1]
    return lst
```

```
# Calculate the total distance of the schedule
def fitness(solution):
    resource = 1
    day = 1
    timeleft = daylength
    totaltime = 0
    distance = 0
    depot = -26.117920, 28.135320
    fromcoords = depot
    deviation = 0
    count = 0

    for i in solution:

        tocoords = coords(i)
        travel = d(fromcoords, tocoords) / speed
        service = serv(i)

        if (travel + service) > timeleft:
            travel = d(fromcoords, depot)
            distance += travel
            totaltime += travel / speed
            deviation += (daylength - totaltime) ** 2
            count += 1

            totaltime = 0
            timeleft = daylength
            fromcoords = depot
            resource += 1
            travel = d(fromcoords, tocoords) / speed

        if resource > employed:
            totaltime = 0
            timeleft = daylength
            fromcoords = depot
            resource = 1
            day += 1
            travel = d(fromcoords, tocoords) / speed

        if (travel + service) <= timeleft and resource <= employed:
            totaltime += travel + service
            timeleft -= travel + service
            distance += travel * speed
            fromcoords = tocoords

    deviation += (daylength - totaltime) ** 2
    standarddeviation = (deviation / (count + 1)) ** (0.5)

    return round(distance, 2), round(standarddeviation, 2)


# Create a population of individuals
def population(solution, size):
    return [random.sample(solution, len(solution)) for i in range(size)]


# Calculate average fitness of a population
def grade(pop):
    dis = 0
    dev = 0
    for i in pop:
        dis += fitness(i)[0]
        dev += fitness(i)[1]
    return round(dis / len(pop), 2), round(dev / len(pop), 2)
```

```
# Tournament selection
def selection(pop, k):
    best = null
    for i in range(1, k):
        ind = pop[random.randint(1, len(pop) - 1)]
        if (best == null) or ind[0] < best[0]:
            best = ind
    return best


# Crossover
def crossover(parent1, parent2):
    l = len(parent1)

    # Get 2 random numbers between 0 and length of individual
    r1 = random.randint(0, l - 1)
    r2 = random.randint(0, l - 1)
    while r1 >= r2:
        r1 = random.randint(0, l - 1)
        r2 = random.randint(0, l - 1)

    # Create the child... initial elements are -1
    child = [-1] * l

    # Copy elements between r1, r2 from parent1 to child
    for i in range(r1, r2 + 1):
        child[i] = parent1[i]

    # Create list to hold elements of parent1 which are not in child yet
    y = [0] * (l - (r2 - r1) - 1)
    j = 0
    for i in range(0, l):
        if parent1[i] not in child:
            y[j] = parent1[i]
            j += 1

    # Order of places is the same as the number of elements after r2
    copy = parent2[:]
    rotate(copy, r2)

    # Order the elements in y according to their order in parent2
    y1 = [0] * (l - (r2 - r1) - 1)
    j = 0
    for i in range(0, l):
        if copy[i] in y:
            y1[j] = copy[i]
            j += 1

    # Copy remaining elements into child according to their order in parent2
starting after r2
    for i in range(0, len(y1)):
        ci = (r2 + i + 1) % l
        child[ci] = y1[i]

    return child


# Mutate
def swap(solution):
    x = random.randint(0, len(solution) - 1)
    y = random.randint(0, len(solution) - 1)
    solution[x], solution[y] = solution[y], solution[x]
    return solution
```

```
# Evolve the population
def evolve(graded, mutate=0.01, retain=0.2):

    p = [x[1] for x in graded]
    retain_length = int(len(p) * retain)
    parents = p[:retain_length]
    parents_length = len(parents)
    desired_length = len(p) - parents_length
    children = []

    while len(children) < desired_length:

        p1 = selection(p, 3)
        p2 = selection(p, 3)


        if p1 != p2:
            child1 = crossover(p1, p2)
            child2 = crossover(p2, p1)
            children.append(child1)
            children.append(child2)

    parents.extend(children)

    for i in parents:
        if mutate > random.random():
            parents.remove(i)
            parents.append(swap(i))

    graded = [((fitness(x)), x) for x in parents]
    graded = [x for x in sorted(graded)]
    fitnesses = [x[0] for x in graded]
    print(fitnesses[0],fitnesses[99])

    return graded


def output(solution):

    resource = 1
    day = 1
    schedule = []
    timeleft = daylength
    totaltime = 0
    traveltime = 0
    servicetime = 0
    FromCoords = depot

    for i in solution:

        ToCoords = coords(i)
        travel = d(FromCoords, ToCoords) / speed
        service = serv(i)

        if (travel + service) > timeleft:
            traveltime += d(FromCoords, depot) / speed
            totaltime += d(FromCoords, depot) / speed

            z = []
            for j in schedule:
                z.append(devices(j))

            print(str(day) + ";" + str(resource) + ";" + str(z) + ";" +
str(totaltime) + ";" + str(traveltime) + ";" + str(servicetime))
```

```
            totaltime = 0
            traveltime = 0
            servicetime = 0
            timeleft = daylength
            schedule = []
            FromCoords = depot

            resource += 1

            travel = d(FromCoords, ToCoords) / speed

        if resource > employed:
            totaltime = 0
            traveltime = 0
            servicetime = 0
            timeleft = daylength
            schedule = []
            FromCoords = depot
            resource = 1
            day += 1

            travel = d(FromCoords, ToCoords) / speed

        if (travel + service) <= timeleft and resource <= employed:
            totaltime += travel + service
            timeleft -= travel + service
            traveltime += travel
            servicetime += service
            schedule.append(i)
            FromCoords = ToCoords

    z = []
    for i in schedule:
        z.append(devices(i))

    print(str(day) + ";" + str(resource) + ";" + str(z) + ";" + str(totaltime) +
";" + str(traveltime) + ";" + str(servicetime))


initial = initialise()

p = population(initial, 100)

pop = [((fitness(x)), x) for x in p]
pop = [x for x in sorted(pop)]

for i in range(100):
    pop = evolve(pop)

print(output(pop[0][1]))
```