Facilities Management: Optimisation of Maintenance Schedules

by

Aubrey Lebese

27386971

Submitted in partial fulfillment of the requirements for the degree of

Bachelors of Industrial Engineering

in the

Faculty of Engineering, Built Environment and Information Technology

University of Pretoria

November 4, 2010

Executive summary

This is the formal document for the project conducted to optimise maintenance schedules for ABC. ABC is a facilities management company that offers total integrated facilities management to its clients. ABC's objective is to enable its clients to concentrate on their core business, which is the business they do best. Fourier was requested by ABC to assist in the optimisation of the master maintenance plans and schedules that are currently used. Maintenance schedules can be optimised by reducing the traveling time to the sites and the number of trips made to each site.

The aim of the project is to develop a scheduling technique that can assist ABC to schedule tasks in such a manner that multiple trips to the same site and traveling time are minimized. A literature review was conducted to find ways in which the aforementioned aim can be achieved. After conducting the research it was decided that the model used for the MTMCPTD should be modified to address ABC's problem. It was also decided that a tabu search heuristic embedded in an adaptive memory program should be employed to solve the problem since the problem is NP-hard.

The tabu search heuristic was formulated using Octave and an instance of the problem was solved to demonstrate the effectiveness of the algorithm. The solution attained bore witness of the fact that improvements can be made to the original schedules by using the formulated algorithm. The improvements made are in terms of reduced traveling time and increased value added time.

Contents

1 Problem Overview **7**
- 1.1 Introduction . . . 7
- 1.2 Problem definition . . . 8
- 1.3 Research design . . . 9
- 1.4 Research methodology . . . 9
- 1.5 Document structure . . . 10

2 Literature review **11**
- 2.1 Approaching a scheduling problem . . . 11
- 2.2 Modelling the problem . . . 12
  - 2.2.1 Multiple Tour Maximum Collection Problem with Time-Dependent rewards (MTMCPTD) . . . 12
  - 2.2.2 Vehicle Routing Problem with Time Windows (VRPTW) . . . 14
  - 2.2.3 Modelling approach to be used . . . 14
- 2.3 Solution approaches . . . 15
  - 2.3.1 Heuristics . . . 15
  - 2.3.2 Metaheuristcs . . . 15
  - 2.3.3 Solution approach to be used . . . 17
- 2.4 Concluding the literature review . . . 17

3 Model formulation **18**
- 3.1 Original formulation of the model . . . 18
  - 3.1.1 Defining subsets . . . 18
  - 3.1.2 Decision variables . . . 19
  - 3.1.3 Inputs for the model . . . 19
  - 3.1.4 Equations for the formulation . . . 20

4 TS Algorithm **21**
- 4.1 General formulation of the TS algorithm . . . 21
- 4.2 Octave program . . . 23
- 4.3 Algorithm verification and validation . . . 24

5 Execution and analysis of results **26**
- 5.1 Algorithm execution . . . 26
- 5.2 Sensitivity analysis . . . 28

6 Conclusion and future research opportunities **30**

Bibliography **30**

A The Octave code **32**

B The reward matrix **48**

C The service time vector **50**

D The traveling time matrix **52**

E Original schedule **55**

F Optimised schedule **58**

List of Figures

2.1 An illusrative example 13

4.1 Octave input 24
4.2 Octave output 25

5.1 Sensitivity of the algorithm to changes in neighborhood size 29

A.1 The Tabu_search_heuristic function (page 1) 33
A.2 The Tabu_search_heuristic function (page 2) 34
A.3 The initiation function 35
A.4 The vertice_insert function 36
A.5 The vertice_insert_2 function 37
A.6 The availability_check function 38
A.7 The availability_check_2 function 39
A.8 The neighborhood function (page 1) 40
A.9 The neighborhood function (page 2) 41
A.10 The store_data function 42
A.11 The tabulist function (page 1) 43
A.12 The tabulist function (page 2) 44
A.13 The evaluate function (page 1) 45
A.14 The evaluate function (page 2) 46
A.15 The improvement function 47

B.1 The reward matrix 49

C.1 The service time vector 51

D.1 The traveling time matrix (page 1) 53
D.2 The traveling time matrix (page 2) 54

E.1 The original schedule 56
E.2 The original schedule for workers 57

F.1 The optimised schedule 59
F.2 The optimised schedule for workers 60

List of Tables

1.1 Weekly schedule 9

4.1 Overview of the tabu search heuristic 21
4.2 Partial solution generation 22
4.3 Solution improvement 22
4.4 Evaluation of neighborhood solutions 23

5.1 Original maintenance schedule for resource/maintenance team 1 26
5.2 Original maintenance schedule for resource/maintenance team 2 26
5.3 Original maintenance schedule for resource/maintenance team 3 27
5.4 Optimised maintenance schedule for resource/maintenance team 1 27
5.5 Optimised maintenance schedule for resource/maintenance team 2 27
5.6 Optimised maintenance schedule for resource/maintenance team 3 28

List of Acronyms

ABC	The facilities management company
GA	Genetic search
MRS	Multi Resource Scheduling
MTMCP	Multiple Tour Maximum Collection Problem
MTMCPTD	Multiple Tour Maximum Collection Problem with Time-Dependent rewards
OEM	Original Equipment Manufacturer
OR	Operations Research
PDP	Pickup and Delivery Problem
SA	Simulated annealing
TS	Tabu search
TSP	Traveling Salesman Problem
VRP	Vehicle Routing Problem
VRPTW	Vehicle Routing Problem with Time Windows

Chapter 1

Problem Overview

1.1 Introduction

Facilities management is an interdisciplinary field primarily dedicated to the maintenance of commercial or institutional buildings, such as schools, hotels, office complexes, sports arenas. The objective of facilities management is to ensure that the facility delivers the core business of the organisation in an efficient manner. The function of facilities management is to ensure safe, secure, and environmentally-sound operations and maintenance of these assets in a cost effective manner aimed at long-term preservation of the asset value. Facilities management is a continuous process of service provision, as it is performed during the operational phase of the life cycle of the building (Wikipedia, 2010).

ABC is a facilities management company that offers total integrated facilities management to its clients. ABC's objective is to enable its clients to concentrate on their core business, which is the business they do best. The aforementioned objective is attained by deriving integrated facilities management strategies that minimises total life-cycle costs and creates and sustains a working environment through a combination and integration of various services that ensure maximum productivity of the clients people and equipment.

The company currently offers integrated facilities management services for 6500 buildings, with a total building area of $\pm 2500000 m^2$, situated on 3300 sites. The sites are geographically dispersed all over South Africa. The sites are grouped into regions and in each region there is a service facility that is responsible for maintaining buildings in that particular region. Different maintenance tasks are performed in each site (e.g electrical, air conditioner etc.).

Fourier Approach (Pty) ltd is a cosulting firm consisting of industrial engineers and computer scientists (Fourier, 2010a). The company prides itself in the alignment of the business application and processes with the strategic direction of its clients (Fourier, 2010b). Many companies have well thought out strategies yet struggle to compete in the market place . This is due to the misalignment of the business processes and strategic direction of the company. Fourier has been working with ABC for a couple of years, initially as part of developing and designing the information strategy and later in the system development, implementation and support areas.

Fourier was recently involved in the examination of SAP Multi Resource Scheduling (MRS) system for ABC. The SAP MRS system is used to schedule maintenance tasks and release job orders. After the examination of the MRS system, it was decided that the maintenance schedules should be optimised in order for the MRS system to add more value to ABC since the maintenance schedules are used as input for the MRS system. It was also realized that a significant improvement can be attained even before the MRS

is implemented. Therefore a decision was taken to commence with the optimisation of maintenance schedules in parallel to completing the customization and changes to the SAP MRS software. Fourier was requested to assist ABC with optimising its existing master maintenance plans and schedules that are used currently. Maintenance schedules can be optimised by reducing the traveling time to the sites and the number of trips made to each site.

1.2 Problem definition

Scheduling of resources is critical in enabling management to predict the future state of the system in terms of resources required to meet customer demand. Scheduling is particularly important for ABC since they offer integrated facilities management to ±3300 sites.

Every piece of equipment/asset under the control of ABC has a maintenance plan linked to it as per specifications of the Original Equipment Manufacturer (OEM). These specific OEM maintenance plans have been refined and adjusted by functional experts from ABC. The maintenance plan contains the following information:

- Interval between subsequent services.
- Types of services and activities that need to be performed.
- Specific resources and grades of resources required.
- Requirements (e.g. A statute that prescribes minimum and maximum window periods).

Currently the maintenance plans for individual equipments/assets under the control of ABC are not integrated so that opportunities to reduce travel time can be exploited. At the moment, the maintenance personnel follow the refined OEM maintenance plans for each equipment/asset. Which means the maintenance personnel will service an equipment/asset on the day that is specified on the OEM maintenance plan. Taking into consideration the fact that many equipments/assets are situated on the same site, this results in a situation where maintenance personnel will typically go to a building and perform the scheduled maintenance task(s) and may then have to go to the same building again in a couple of days or a week or two. This clearly causes ineffective use of resources in a sense that, more time is spend on traveling rather than performing value adding activities. This ineffectiveness leads to high operational cost in a form of fuel cost and maintenance cost for vehicles being used. The utilisation of resources is also lowered as resources spent more time being unproductive.

A typical scenario is shown in Table 1.1. From the table it can be seen that site 1, 2, and 3 will be visited twice in one week. The aim of the project is to devise a plan that can assist ABC to schedule tasks in such a manner that multiple trips to the same site are minimized.

In order to complete the project successfully, certain issues must be resolved. The research question that we ask in this project is:

> *"Which maintenance tasks should be performed on a specific day to reduce traveling time?"*

Table 1.1: Weekly schedule

Site to visit	**Monday**	**Tuesday**	**Wednesday**	**Thursday**	**Friday**
Site 1	1	0	1	0	0
Site 2	0	1	0	0	1
Site 3	0	0	0	1	1
Site 4	1	0	0	0	0
Site 5	0	0	0	1	0
Site 6	0	0	0	0	1
Total sites to visit	**2**	**1**	**1**	**2**	**3**

1.3 Research design

The purpose of the project is to provide an answer to the aforementioned research question. An exhaustive research is conducted in Chapter 2 in order to find an effective technique that can be used to solve the problem. The research question can be answered by developing and solving a model that can be used to optimise maintenance schedules. The following deliverables will be presented after completion of the project:

1. The mathematical model that can be used to solve the problem.
2. The solution algorithm to the model.
3. A solution to an instance of the mathematical model.
4. A sensitivity analysis on the optimal solution attained.

1.4 Research methodology

The problem being studied falls under the category of problems that require the application of operations research (OR) to be solved. OR is a scientific approach to decision analysis that is aimed at finding the optimal design and operate a system, given a limited number of resources (Winston and Venkataramanan, 2003). Rajgopal (2004) propose seven steps for an OR study. The focus of this project will only be on step 2-6.

1. **Orientation**: The first step in an OR study is to identify and document the problem or inefficiencies that need to be addressed.
2. **Problem Definition**: The second step encompasses the definition of the organisational problem. This involves defining the objective of the organisation and stating the parts of the organization which need to be studied in order to solve the problem.
3. **Data collection**: During this step the operations researcher needs to collect the necessary data required to solve the problem. The data is in the form of estimates of the parameters that have an effect on the organization's problem. The collected data will be used to develop and evaluate the model.
4. **Model Formulation**: In this step, a fully functional model is developed to solve the problem. The objective of the model is minimize traveling time while ensuring that other business requirements are still met.

5. **Solution**: In this step, a method is selected to solve the problem. An exact solution can be attained by using algorithms or an optimisation software if the problem is not NP-complete. Heuristics will be used to solve the problem if it is NP-complete.

6. **Validation & output analysis**: In this step, the model is analysed to check if it is an accurate representation of reality. The decision values yielded by the model should be consistent with reality. A sensitivity analysis is then performed in order to analyse the sensitivity of the output to changes in the values of input parameters used.

7. **Implementation and monitoring**: The attained solution is then implemented and monitored. The results attained from the implementation of the solution are then compared with the expected results to see if the solution brings about improvements.

1.5 Document structure

The remaining part of the document will be organised as follows. A literature review on the problem will be given in the next chapter. The aim of conducting a literature review is to find a strategy that can be used to model and solve the problem. In Chapter 3 the model that will be used to solve ABC's problem is presented. In Chapter 4 the algorithm that will be used to solve the problem is developed. In Chapter 5 an instance of the problem is solved and a sensitivity analysis is performed on the attained solution. Finally Chapter 6 concludes the project and discusses future research opportunities.

Chapter 2

Literature review

2.1 Approaching a scheduling problem

ABC is confronted with a task scheduling problem. The scheduling process is concerned with the allocation of resources between a variety of tasks which require the same resources for completion. In this particular instance, ABC has to decide on the sequence in which the sites must be visited by maintenance personnel for maintenance. The sequence must result in minimum travel time and maximum resource utilisation while taking customer satisfaction into consideration. Customer satisfaction is constituted by service date of the equipment/asset. Servicing the equipment/asset before the due date specified in the OEM maintenance plan results in higher customer satisfaction while a lower customer satisfaction is attained for servicing the equipment/asset after the due date.

There are several modelling approaches that can be utilised to solve the problem (Coyle et al., 2003), which are:

- Simulation models
- Heuristic optimisation models
- Optimisation models

The problem will be modelled as an optimisation model, due to the fact that other modelling approaches give an approximation of the optimal solution while optimisation models find the exact solution provided the problem is not NP-complete. An NP-complete problem is a problem which requires excessive amount of time to be solved. According to Winston and Venkataramanan (2003), a problem is said to be NP-complete if there is no algorithm that can solve large instances of the problem in polynomial time. Such problems are solved using heuristic techniques. The scheduling problem being analysed falls under an umbrella of problems known as vehicle routing and scheduling problems. Four types of problems that fall under the aforementioned category of problems were identified during the research, which are:

- Traveling Salesman Problem (TSP).
- Vehicle Routing Problem (VRP).
- Pickup and Delivery Problem (PDP).
- Multiple Tour Maximum Collection Problem (MTMCP).

Other problems are developed from the above mentioned problems to meet specific organisational requirements that are not met by the those standard problems. The time window factor is one variant that is mostly added to the standard problems, and can be defined as the period in which a task can be performed.

2.2 Modelling the problem

The first step in modelling the problem is concerned with identifying the class of problem being dealt with. ABC's problem best fits the following two classes of problems:

1. Multiple Tour Maximum Collection Problem with Time-Dependent rewards (MTMCPTD).
2. Vehicle Routing Problem with Time Windows (VRPTW).

2.2.1 Multiple Tour Maximum Collection Problem with Time-Dependent rewards (MTMCPTD)

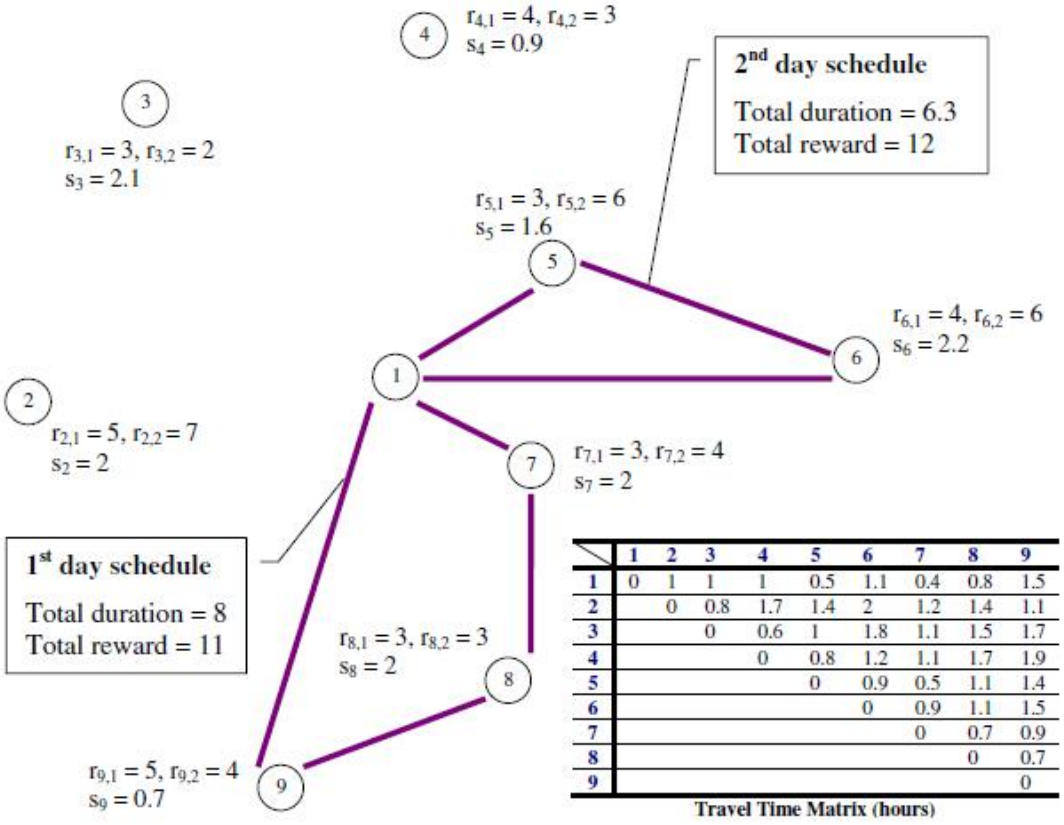
The objective of the MTMCPTD, as defined by Tang et al. (2007), is to determine a set of tours, each corresponding to the maintenance personnel's schedule on a particular day, such that the total reward collected during the planning horizon is maximised. The MTMCPTD is the dynamic version of the multiple tour maximum collection problem (MTMCP) (Tang et al., 2007). The only difference between MTMCPTD and MTMCP is that the rewards remain constant throughout the planning horizon in a MTMCP while the rewards change as a function of time in a MTMCPTD. By maximising the total reward collected, the utilisation of resources will be maximised and traveling time will be minimized. This is due to the fact that greater reward can only be collected by dedicating less time to traveling.

The reward collected for completing a particular task is dependent on the day the task was completed. A reward is based on the level of inconvenience that will be caused by taking the equipment/asset out of the system for maintenance. For instance, taking a stove used for cooking in a restaurant out for maintenance during peak hours is more costly than taking it out during off-peak hours. Greater reward is allocated to tasks near their due date, so that the tasks can be selected. Rewards are not in monetary terms, rather, they are internal values used by the algorithm to schedule tasks in an appropriate manner. The tour duration should not exceed the working hours of the maintenance personnel and a tour should begin and end at the regional offices. The input parameters for the MTMCPTD are:

- The distance between the nodes (all maintenance sites and the regional office are defined as nodes).
- The travel time between the nodes.
- Time required to complete a task at each site.
- The due date for each task.
- The time dependent rewards associated with each task

ABC's problem has similar characteristics to that of the MTMCPTD discussed above. In both problems the utilisation of resources is being maximised and the traveling time is being minimised. There are a couple of differences between ABC's problem and the MTMCPTD, which are:

- In the MTMCPTD the time required to complete a task cannot be greater than the working hours per day, while in ABC's problem tasks can require more than the working hours to complete.
- In the MTMCPTD a single tour is determined for each working day, while in ABC's problem multiple tours should be determined each day since there are more than one maintenance workers.



	1	2	3	4	5	6	7	8	9
1	0	1	1	1	0.5	1.1	0.4	0.8	1.5
2		0	0.8	1.7	1.4	2	1.2	1.4	1.1
3			0	0.6	1	1.8	1.1	1.5	1.7
4				0	0.8	1.2	1.1	1.7	1.9
5					0	0.9	0.5	1.1	1.4
6						0	0.9	1.1	1.5
7							0	0.7	0.9
8								0	0.7
9									0

Figure 2.1: An illusrative example

Figure 2.1 shows an illustrative example of the scheduling problem with one technician and the planning horizon of two days(Tang et al., 2007). The reward for each task/vertex is indicated on the figure adjacent each vertex as:

$$r_{ik} \quad \forall \quad i \in \{1, \ldots, 9\}, k \in \{1, 2\}$$

The service time is also indicated on the figure adjacent each vertex as:

$$s_i \quad \forall \quad i \in \{1, \ldots, 9\}$$

The traveling time matrix is shown on the table at the bottom. The matrix depicts the traveling time between all the vertices. The optimal solution for the problem is shown on the figure by connecting the vertices that make-up the solution.

The aforementioned differences will have to be incorporated in the model used for the MTMCPTD so that the requirements for ABC's problem can be met. The model used for the MTMCPTD can be criticized for its subjectiveness in assigning rewards to tasks.

2.2.2 Vehicle Routing Problem with Time Windows (VRPTW)

The VRPTW is the generalization of the vehicle routing problem (VRP) where service at any customer can take place within an allocated time period called the time window. According to Vacic and Sobh (2002) the objective of a VRP is to find a set of routes, starting and ending at the depot/regional offices, that will cover a set of customers, such that the total traveling distance or number of vehicles is minimised, or a combination of these. The objective of the VRPTW is the same as that of the VRP. The time windows are considered hard when they cannot be violated and soft when they are non-binding (Goel and Gruhn, 2006).

In a VRPTW, vehicles are dispatched from the central node to the customers in order to fulfill the demand associated with each customer. The time window corresponds to the period when the customer is available to accept the order (Solomon, 1985). All the routes start and end at the central node and each route represents a single tour that will be taken by one specific vehicle. The input parameters for the VRPTW are:

- The distance between the nodes (nodes include customer nodes and the central node).
- The travel time between the nodes.
- Time required to complete fulfill an order for each customer.
- The time window to service each customer.
- Demand for each customer.
- Capacity for each vehicle

There is some resemblance between ABC's problem and the VRPTW. The objective of both problems is to minimize traveling time. There are a couple of differences between the VRPTW and ABC's problem, which are:

- The VRPTW does not consider multiple working days.
- A site can only be visited once in the VRPTW.

2.2.3 Modelling approach to be used

From the literature provided of the two classes of problems discussed above, it can be concluded that the model used for the MTMCPTD requires less changes compared to the model used for the VRPTW for it to be used to address ABC's problem. Therefore the model used for the MTMCPTD will be modified to address ABC's problem. The modifications that have to be added are:

- The model should be modified to cater for tasks that have a service time that is greater than the working hours in a day.
- Multiple tours should be scheduled for each day since there are more than one maintenance workers.

2.3 Solution approaches

The objective of any operations research project is to solve the formulated model. The first option that comes into the mind of an operations researcher involved in project is to use the exact method to solve the problem. The extent to which a model can be solved using the exact method depends on the complexity and size of the model. But the first option is not feasible in some cases. Therefore other methods to solve the problem should be utilized (Rajgopal, 2004)

The MTMCPTD, which is the dynamic version of the MTMCP, is the problem class that was identified as fitting the problem being analysed. Butt and Cavalier (1994) discovered that the MTMCP is NP-hard, which means large instances of this problem cannot be solved by using an exact method. The problem is considered to be large when it contains more than 20 vertices that have to be visited. Therefore it can also be said that the MTMCPTD is NP-hard since it is the dynamic version of the MTMCP. The problem can be solved by the use of heuristics and metaheuristics

2.3.1 Heuristics

Heuristics are applied to problems which are difficult to solve (NP-complete or NP-hard). Heuristics are techniques used to approximate the solution of an optimisation model (Winston and Venkataramanan, 2003). Techniques which do not guarantee optimality when used are called heuristics. Those techniques employ a greedy approach to arrive at a good solution in an efficient time. Heuristics tend to be trapped at the local optimal and fail to find the global optimal solution due to the greedy nature they employ to find solutions (Winston and Venkataramanan, 2003).

2.3.2 Metaheuristcs

Metaheuristics are based on artificial intelligence and are general master strategies to solve problems. Unlike heuristics, metaheuristics try to avoid being trapped at the local optimal solution by accepting solutions that may not lead to improvement. They are also able to terminate the algorithm without cycling. The following three categories of metaheuristics will be discussed:

- Genetic search (GS).
- Simulated annealing (SA).
- Tabu search (TS).

Genetic search (GA)

Genetic algorithms (GA) are based on the principle of evolution. GA was developed by Holland (1975) based on the genetic principles as a methodology to solve problems in the decision-analysis field. Winston and Venkataramanan (2003) proposed the following conventional steps that are followed in any genetic algorithm:

Step 1: *Generation* of an initial population solution. The size of the population is set to a predetermined value of n, and random values are generated. This is called generation 1.

Step 2: *Evaluation*. Score each solution on the basis of a fitness function.

Step 3: *Selection.* Choose probabilistically the parents for the next generation from the current solution.

Step 4: *Reproduction.* Crossover the parents at a random point on the gene string in order for the offstring to have a portion of each parent.

Step 5: *Mutation.* Alter the genetic makeup to avoid being trapped at the local optima.

Step 6: *Repeat* steps 2 through 5 until an optimal solution is reached.

The algorithm starts by randomly generating an initial population of strings called the first generation in step 1. Each string represents a possible solution which is evaluated by using a measuring criteria in step 2. The measuring criteria is used to assign a fitness value to each of the generated strings. The next generation is generated by mating the previous generation and performing some mutations. The mating partners are selected based on their fitness values in step 3. The higher the fitness value of a string, the more chances it has to be choosen to reproduce in step 4. In step 5 the make-up of the the offspring is altered and it becomes the new generations solution.

Simulated annealing (SA)

Simulated annealing (SA) is based on the behaviour of a set of atoms in achieving thermal equilibrium at a certain temperature. This algorithm was developed by Metropolis et al. (1953) and it emulates the process of aggregating particles in a physical system as it is cooled. When the temperature of the system is slowly lowered, the energy exchange allows for true equilibrium in each stage until the global minimum energy is reached. The algorithm accepts solutions that do not result in an improved state with a certain probability to avoid being trapped at the local equilibrium points. The initial temperature is called the melting temperature and the steps in lowering the temperature are called cooling schedule. The technique is memory-less since it optimizes without prior knowledge of the solution strategy. The algorithm consists of three basic steps (Winston and Venkataramanan, 2003), which are:

Step 1: Generate the initial solution $\boldsymbol{x}_0$ for the problem. Determine the initial temperature T_0. Set the initial parameters for the algorithm. Set the number of iterations I, cooling schedule temperature reduction δ, $T_{cur} = T_0$, $x_{cur} = \boldsymbol{x}_0$, $x_{best} = x_{cur}$ and $z_{best} = z(x_{cur})$.

Step 2: For all the iterations from $count = 1$ to I, generate a new solution $\boldsymbol{x}_{count}$. If the objective function value $z(\boldsymbol{x}_{count})$ is better than the current objective function value $z(x_{cur})$ set $x_{cur} = \boldsymbol{x}_{count}$. If the objective function value $z(\boldsymbol{x}_{count})$ is worse than $z(x_{cur})$, then calculate $\Delta C = z(\boldsymbol{x}_{count}) - z(x_{cur})$ and set $x_{cur} = \boldsymbol{x}_{count}$ with a probability of $e^{(\Delta C/T_{cur})}$. If z_{best} is worse than $z(x_{cur})$, then set $x_{best} = x_{cur}$, and $z(best) = z(x_{cur})$. Update the value of the iteration number *count*.

Step 3: Set $T_{cur} = T_{cur} - \delta$. If the stopping criteria or another predetermined stopping criteria is satisfied, then stop and output z_{best} and x_{best} as the solution. Otherwise, return to step 2, with T_{cur}.

Preliminary studies have to undertaken to estimate the values for δ and I.

Tabu search (TS)

The Tabu search (TS) was developed by Glover (1986) based on the strategies employed in intelligent decision making. TS uses short- and long-term memory to avoid certain moves in the solution space. The short-term memory helps in restraining the algorithm from cycling around the local neighborhood in the solution space. It also helps to move away from the local optimal solution. Long-term memory helps the algorithm to search in promising neighborhoods. TS does not emulate any physical or biological process but makes extensive use of memory. The forbidden moves are stored in the tabu list and each iteration chooses a non-tabu feasible solution. After completion of each step, all the moves that will result in returning immediately to the previous solution are stored in the tabu list. The moves stored in the tabu list are not allowed for a couple of iterations, but they are eventually removed from the tabu list and again available for selection. The algorithm consists of three basic steps (Rardin, 1998), which are:

Step 1: Generate the initial solution $\boldsymbol{x}_0$. Set the maximum number of iterations t_{max}. Set the best solution $x_{best} = \boldsymbol{x}_0$. Set the neccessary parameters. Set short-term memory size, tabu criteria, and neighborhood size.

Step 2: For all the iterations from $t = 1$ to t_{max}, generate a new solution $\boldsymbol{x}_t$. Choose some non-tabu feasible move $\Delta\boldsymbol{x}$ as $\Delta\boldsymbol{x}_{t+1}$. Update $\boldsymbol{x}_{t+1}$ as $\boldsymbol{x}_{t+1} = \boldsymbol{x}_t + \Delta\boldsymbol{x}_{t+1}$. If the objective function value of $\boldsymbol{x}_{t+1}$ is superior to that of x_{best}, replace $x_{best} = \boldsymbol{x}_{t+1}$. Remove the forbidden moves that have been in the tabu list for a sufficient number of iterations, and add a collection of moves that will result in returning immediately from $\boldsymbol{x}_{t+1}$ to $\boldsymbol{x}_t$.

Step 1: If non-tabu move Δx leads to a feasible neighbor of the current solution $\boldsymbol{x}_t$, or if $t = t_{max}$, then stop. Output x_{best} as the solution for the algorithm.

2.3.3 Solution approach to be used

A tabu search heuristic will be employed to solve the problem since Tang et al. (2007) showed that the heuristic is able to provide near optimal solution for large-size problems. The ability of the tabu search heuristic to make extensive use of memory gives it an extended advantage over other metaheuristic techniques. The other advantage that the tabu search heuristic has over other techniques is its ability to solve large-size models.

2.4 Concluding the literature review

From the literature review conducted one can conclude that ABC's problem can be modeled by modifying the model used for the MTMCPTD, since the problems are similar in most aspects. The model cannot be solved by the use of the exact method since it is complex and large in size. Therefore the model can be solved by employing a tabu search heuristic.

Chapter 3

Model formulation

In this chapter a mathematical model of ABC's problem will be formulated to better understand and eventually solve the problem. As it was concluded in the previous chapter, the problem will be modeled as a MTMCPTD. According to Tang et al. (2007), the the objective of the MTMCPTD is to determine a set of tours, each corresponding to a technicians schedule on a particular day, such that the total reward collected during the scheduling horizon is maximized. For the purposes of the model, a couple of assumptions had to be made to prevent the model from being cumbersome. The following assumptions were made:

1. The task duration of any task should be less than the total working time per day.
2. A tour should begin and end at the origin/regional offices.
3. A task should be completed in the same day that the work was started on that specific task.
4. The total working time per day does not include breaks taken by workers to eat or attend to any personal matter.
5. Unscheduled maintenance will not be considered in the scheduling process

3.1 Original formulation of the model

3.1.1 Defining subsets

Due to the numerous grouping of tasks that are needed to solve the problem, the introduction of subsets becomes important in the formulation of the model. The subsets that will be defined are all formed from the vertex set below:

$$\boldsymbol{V} = \{1, 2, 3, \ldots, n\}$$

A single vertex represents a location where the is a task that has to be performed. The set above consists of all the vertices that have to be visited including the central depot/regional offices. The central depot/regional offices is defined as vertex number one. The second set that will be used in the formulation of the model is the one representing the planning horizon. The set is denoted as follows:

$$\boldsymbol{H} = \{1, 2, 3, \ldots, m\}$$

The first subset represents all the vertices except for the central depot/regional offices. The subset is denoted as follows:

$$\boldsymbol{V^C} = \boldsymbol{V}\backslash\{1\}$$

The second subset represents a group of vertices that will be visited in a particular day during the planning horizon. The subset is denoted as follows:

$$\boldsymbol{U} \subset \boldsymbol{V}$$

3.1.2 Decision variables

The goal of the MTMCPTD is to determine a set of tours to be undertaken in the period extending until the end of the planning horizon. Thus the decision variable is denoted as follows:

$$y_{ik} \triangleq \begin{cases} 1 & \text{if task } i \text{ is serviced on day } k, \forall\ i \in \boldsymbol{V^C} \text{ and } k \in \boldsymbol{H} \\ 0 & \text{otherwise} \end{cases}$$

The aforementioned output only gives the days in which each task is going to be performed. This gives rise to the need to know in what sequence will the tasks be performed in each day. Therefore, the second output indicates the order in which tasks will be performed in the specific day. The decision variable is denoted as follows:

$$x_{ijk} \triangleq \begin{cases} 1 & \text{if task } j \text{ is serviced immediately after task } i \text{ on day } k, \forall\ i \in \boldsymbol{V^C}, \\ & k \in \boldsymbol{H}, j \in \boldsymbol{V}\backslash\{i\} \\ 0 & \text{otherwise} \end{cases}$$

3.1.3 Inputs for the model

Similar to other optimisation models, the necessary data has to be made available to the model in other for the problem to be solved. Three set of inputs are required to solve the problem, which are:

1. The service time
   A vector containing the service time for each task should be made available to solve the problem. The input is denoted as:

   $$s_i \qquad \forall\ i \in \boldsymbol{V}$$

2. The time-dependant reward for each task
   A matrix containing time-dependant rewards for perform each task on a particular day during the planning horizon.The input is denoted as follows:

   $$r_{ik} \qquad \forall\ i \in \boldsymbol{V}, k \in \boldsymbol{H}$$

3. The traveling time matrix
   A matrix containing the traveling between of all the vertices. The input is denoted as follws:

   $$t_{ij} \qquad \forall\ i, j \in \boldsymbol{V}$$

3.1.4 Equations for the formulation

The objective of the model is to maximise the total reward collected during the planning horizon. Hence, the objective function is formulated as follows in Equation (3.1).

$$\max z = \sum_{i \in \boldsymbol{V^C}} r_{ik} y_{ik} \tag{3.1}$$

subject to

$$\sum_{i \in \boldsymbol{V^C}} x_{1ik} = \sum_{i \in \boldsymbol{V^C}} x_{i1k} = 1 \qquad \forall\ k \in \boldsymbol{H} \tag{3.2}$$

$$\sum_{j \in \boldsymbol{V} \setminus \{i\}} x_{ijk} = \sum_{j \in \boldsymbol{V} \setminus \{i\}} x_{jik} = y_{ik} \qquad \forall\ i \in \boldsymbol{V^C}; k \in \boldsymbol{H} \tag{3.3}$$

$$\sum_{i \in \boldsymbol{V}} \sum_{j \in \boldsymbol{V}} (t_{ij} + s_i) x_{ijk} \leq T \qquad \forall\ k \in \boldsymbol{H} \tag{3.4}$$

$$\sum_{k \in \boldsymbol{H}} y_{ik} \leq 1 \qquad \forall\ i \in \boldsymbol{V^C} \tag{3.5}$$

$$\sum_{i \in \boldsymbol{U}} \sum_{j \in \boldsymbol{U} \setminus \{i\}} x_{ijk} \leq |\boldsymbol{U}| - 1 \qquad \forall\ \boldsymbol{U} \subset \boldsymbol{V^C}; n - 1 \geq |\boldsymbol{U}| \geq 2; k \in \boldsymbol{H} \tag{3.6}$$

$$x_{ijk} \in \{0, 1\} \qquad \forall\ i, j \in \boldsymbol{V}; i \neq j; k \in \boldsymbol{H} \tag{3.7}$$

$$y_{ik} \in \{0, 1\} \qquad \forall\ i \in \boldsymbol{V^C}; k \in \boldsymbol{H} \tag{3.8}$$

Equation (3.2) ensures that each tour starts and ends at the depot. Equation (3.3) guarantees the connectivity of the service tour for each day. Equation (3.4) ensures that each tour that is taken during the planning horizon is less than the predetermined tour duration length (T). Equation (3.5) ensures that all maintenance tasks can be scheduled at most once during the planning horizon. Equation (3.6) prevent sub-tours that do not include the central depot. Equations (3.7) and (3.8) provide for the integrality requirements.

The original formulation shown above can also be used for the problem due to the similarities between the MTMCPTD and the problem at hand.

Chapter 4

TS Algorithm

As it was indicated in the literature review that the MTMCPTD is NP-hard and therefore heuristic techniques will be employed to solve the problem. As a result of the numerous advantages that metaheuristics have over heuristics techniques, a metaheuristic technique was chosen to solve the problem. The chosen metaheuristic technique is the Tabu Search algorithm (TS). The remainder of this chapter will focus on the formulation of the TS algorithm.

4.1 General formulation of the TS algorithm

According to Tang et al. (2007), the tabu search heuristic offers better solutions when it is embedded in an Adaptive Memory Procedure (AMP) than when used exclusively. Hence, a TS heuristic embedded in an AMP is proposed to solve the problem. Table 4.1 shows the general structure of a TS heuristic embedded in an AMP. The AMP processes are performed in Steps 1,2 and 4 while the TS processes are performed in Step 3.

Table 4.1: Overview of the tabu search heuristic

Step 1.	(Vehicle tour generation in AMP) Generate m vehicle tours and store them in the adaptive memory. Let the main iteration index $q_M = 0$.
Step 2.	(Solution construction within AMP) Set $q_M = q_M + 1$. Construct a new solution S with the m tours generated in the previous step in the adaptive memory.
Step 3.	(Improvement of solution S)
Step 4.	(Tour update within AMP) Update the solution tours maintained in adaptive memory. If the main iteration index $q_M < \lambda$, go to Step 2; otherwise, output the best solution and stop.

In Step 1 of the TS heuristic m vehicle tours are generated and stored in the AMP. Table 4.2 gives a detailed description of how Step 1 of the algorithm is carried out. In Step 1A, m tours are created with zero duration (i.e. starting and ending at vertex 1) and all the tasks that can be performed in one working day are stored in the vector $\mathbf{\Omega}$. Steps 1B until 1D all have to do with inserting as many vertices into the m created tours as possible. When no more vertices can be inserted into all of those m tours then the

algorithm moves to Step 2.

Table 4.2: Partial solution generation

Step 1A.	Create m intial tours $\boldsymbol{\tau}_k = \{1, 1\}$ and set their duration such that $D(\boldsymbol{\tau}_k) = 0, (k = 1, 2, \ldots, m)$. Let $\boldsymbol{\Omega} = \{j \vert t_{1j} + t_{j1} + s_j \leq T, \forall\ j \in \boldsymbol{V} \backslash \{1\}\}$. Let $\boldsymbol{H} = \{1, 2, \ldots, m\}$.
Step 1B.	For each $k \in \boldsymbol{H}$, randomly select and remove a vertex j from set $\boldsymbol{\Omega}$, and insert it in $\boldsymbol{\tau}_k$.
Step 1C.	Select $j \in \boldsymbol{\Omega}$ and two vertices p and q on $\boldsymbol{\tau}_k$(where $k = \{t \vert maxr_{jt}\}, \forall\ t \in \boldsymbol{H}$) such that $D(\boldsymbol{\tau}_k) + t_{pj} + t_{jq} - t_{pq} + s_j \leq T$. Otherwise, insert j between the selected p and q, remove j from $\boldsymbol{\Omega}$ and update the duration of $\boldsymbol{\tau}_k$.
Step 1D.	If $\boldsymbol{H} = \varnothing$, stop; otherwise, go to Step 1C.

Step 2 of the algorithm creates an initial solution to be used in Step 3 of the algorithm. The m generated tours in Step 1 are saved in the AMP as the initial solution. Step 3 of the algorithm which performs the actual TS processes is shown in Table 4.3. In this step the initial solution generated is improved by both the greedy and probabilistic approaches. Step 3A and 3B are concerned with generating neighborhood solutions for the initial solution. The neighborhood solutions are generated by randomly choosing two tours from the initial solution and then removing a specified number of vertices from the two tours. Then vertices are randomly selected from the vector containing unperformed tasks and then inserted into the two selected tours until the is no longer any vertex that can be inserted. In Step 3C the generated neighborhood solutions are evaluated and the tabu parameters are adjusted.

Table 4.3: Solution improvement

Step 3A.	Given current solution S, set tabu parameters to a small neighborhood stage in which only a small number of neighborhood solutions will be explored. Let the tabu iteration index $q_T = 0$.
Step 3B.	Set $q_T = q_T + 1$. Generate by both random and greedy procedures a number of neighborhood solutions (feasible and infeasible) to the current solution S based on the current tabu parameters. Note that in select iterations, the sequence of selected neighborhood solutions will be improved by heuristic procedures.
Step 3C.	Select the best non-tabu solution based on quasi-reward values from the candidates generated in Step 3B (tabu status can be overridden if the best tabu solution is better than the current best feasible solution). This solution will override the current solution stored in S. Depending on the current neighborhood size and solution quality, set heuristic parameters

Table 4.4 gives a detailed description of Step 3C. The quasi reward is calculated for each neighborhood solution generated. The quasi reward is a technique used to calculate the reward of the solution while taking into account the infeasibility of the generated

solution. Infeasible solutions are generated in order to avoid being trapped on the local optimal solution. Therefore the quasi reward subtracts the penalty factor which is charged to all infeasible solutions from the original tour reward. The neighborhood solutions are then compared by the use of the calculated quasi reward value. Determine the maximum reward for both the non-tabu (i.e. it's not contained in the tabu list) and tabu solutions (i.e. it's contained in the tabu list). Adjust the intial solution, best solution and best feasible solution as indicated in Table 4.4. The two tabu parameters that need to be adjusted are the neighborhood size(β) and the penalty coefficient(η).

Table 4.4: Evaluation of neighborhood solutions

Step A.	For each generated neighborhood solution $\tilde{S}$, calculate the quasi reward using the following expression: $$R^{quasi}(\tilde{S}) = \sum_{k=1}^{m} R(\boldsymbol{\tau}) - \eta \sum_{k=1}^{m} \max(D(\boldsymbol{\tau}) - T, 0)$$
Step B.	With the aid of the tabu list, determine the maximum reward for both the non-tabu and tabu solutions (denoted as S_{nt} and S_t, respectively) out of all neighborhood solutions formulated, based on their quasi rewards. If the total reward of the tabu solution S_t is greater than both the rewards of non-tabu solution S_{nt} and the best solution S^* from all previous iterations of the TS procedure, then let $S^* = S_t, S = S_t$ and $q_T = 0$; otherwise, let $S = S_{nt}$. If $S_{nt} > S^*$, set $S^* = S_{nt}$ and $q_T = 0$
Step C.	If S is feasible and it's greater than the best feasible solution denoted as S_f^*, then set $S_f^* = S$ and $q_T = 0$. Adjust the penalty coefficient η and the neighborhood size parameter β

4.2 Octave program

The algorithm that was coded on octave is shown in Appendix A. The algorithm consists of 11 functions that are embedded in one main function called Tabu_search_heuristic. To run the algorithm, all the functions should be stored in one directory together with the text documents of the required data. The required data as mentioned in the preceding chapters is the service time,reward and traveling time. The service time vector should be stored in a text document with file name `service_time.txt` while the reward matrix should be stored in a text document with file name `reward.txt`. The traveling time matrix should also be stored in text document with file name `traveling_time.txt`. The algorithm can then be summoned by using the command:

Tabu_search_heuristic(T,m)

Where T represents the total working time per day and m represents the planning horizon. Figure 4.1 shows an octave screen where the algorithm is being summoned. The output for the algorithm is an array called *Solution* that contains a number of schedules for

the planning horizon. The number of schedules contained in the array is dependant on the number of tasks that need to be scheduled. A single schedule consists of m tours that need to be taken by one team of maintenance personnel in the entire planning horizon. The higher the number of tasks, the more resources/maintenance teams required to perform the maintenance tasks. The other output is a vector called *Solution_reward* that contains the reward collected for each of the schedules contained in the array *Solution*.

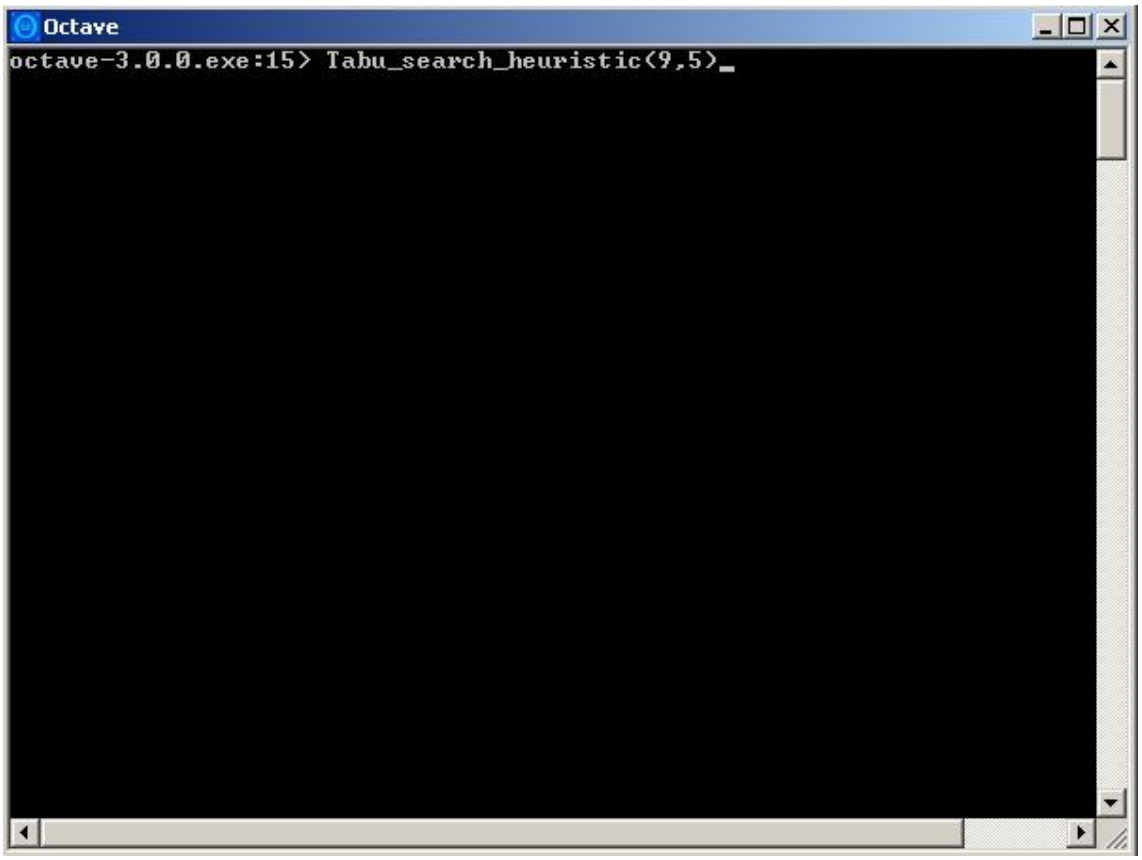


Figure 4.1: Octave input

Figure 4.2 shows the octave output screen. From the figure it can be seen that three schedules have been formulated. Which means three resources/maintenance teams will be required to perform the maintenance tasks. The rows of a single schedule represents a single tour that has to be taken on a specific day. Row 1 represents a tour that has to be taken on day one of the planning horizon, row 2 represents a tour that has to be taken on day two of the planning horizon, and so on. It can also be seen that in any schedule all the tours begin and end with node 1. Node 1 represents the depot/regional office and other nodes/numbers that follow after the first component of a row (node 1) in each row are tasks that have to be performed in the given order. The number/node zero is used to ensure that all the matrix components are defined. Therefore the number/node zero can be considered as an empty component.

4.3 Algorithm verification and validation

It is critical that the algorithm is functioning correctly and is a reasonable representation of reality. In the algorithm building process the algorithm was continually verified and

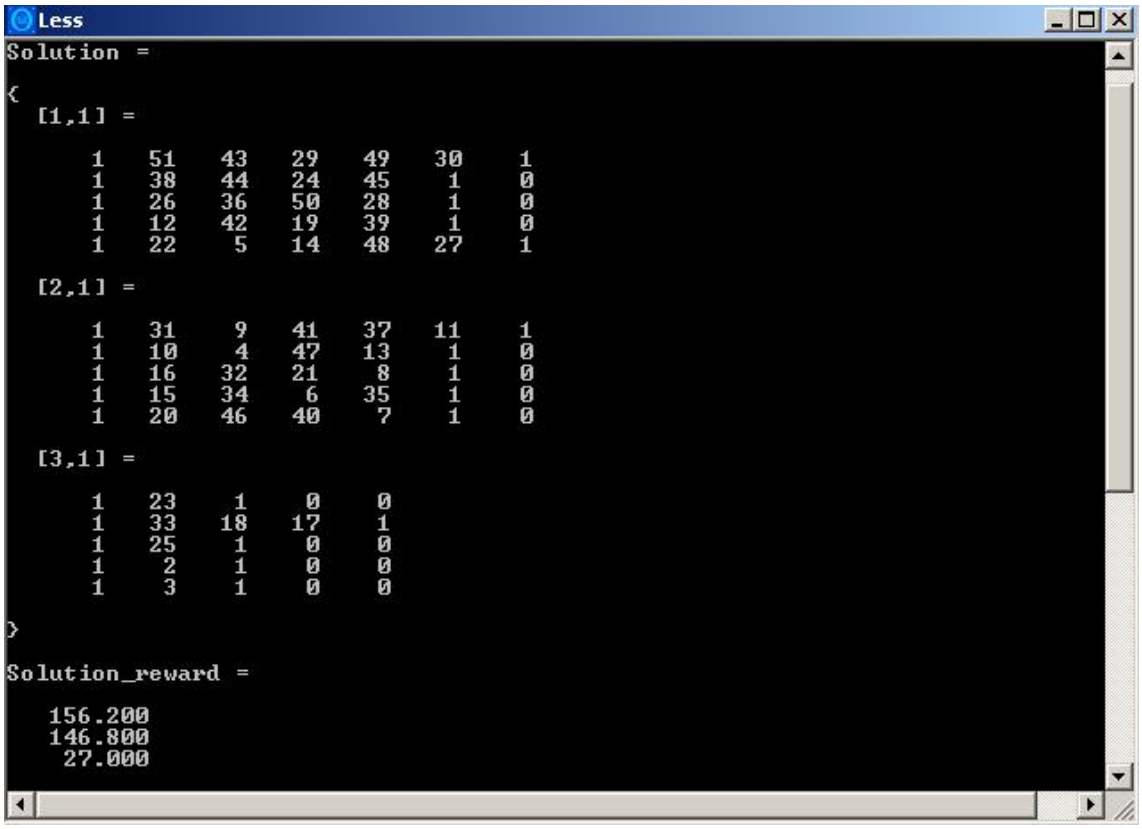


Figure 4.2: Octave output

validated as it was build to enable easier debugging of the code. If an error is discovered after bulding a specific part of the algorithm, the error is corrected before the next part is being build. The answers of the algorithm where checked if they are logical and possible. For instance, the duration of a single tour should not be greater than the working hours per day.

Chapter 5

Execution and analysis of results

5.1 Algorithm execution

The algorithm was used to solve a problem with 50 tasks that needed to be scheduled for the planning horizon of one week (i.e five days). The original scheduling of the tasks is shown in Appendix E. The summarised original schedules for resource/maintenance team 1,2 and 3 are shown in Table 5.1,5.2 and 5.3 respectively. The total percentage value addeded time for resource/maintenance team 1,2 and 3 is 49.7%,46.98% and 41.69% respectively. The total percentage value added time for all the resources is 46.50%.

Table 5.1: Original maintenance schedule for resource/maintenance team 1

	Mon	**Tue**	**Wed**	**Thur**	**Fri**
Total tasks to be performed	3	4	4	4	4
Total traveling time(hours)	4.95	4.75	4.6	3.95	4.3
Total servicing time(hours)	3.4	4.5	4.9	5.3	4.2
Total working time(hours)	8.35	9.25	9.5	9.25	8.5
% value added time	**40.72%**	**48.65%**	**51.58%**	**57.30%**	**49.41%**

Table 5.2: Original maintenance schedule for resource/maintenance team 2

	Mon	**Tue**	**Wed**	**Thur**	**Fri**
Total tasks to be performed	3	3	4	4	4
Total traveling time(hours)	3.47	4.25	4.81	5.05	4.65
Total servicing time(hours)	2.7	3.6	4.2	4.5	4.7
Total working time(hours)	6.17	7.85	9.01	9.55	9.35
% value added time	**43.76%**	**45.86%**	**46.61%**	**47.12%**	**50.27%**

The optimised schedule is shown in Appendix F. The summarised optimised schedules for resource/maintenance team 1,2 and 3 are shown in Table 5.4,5.5 and 5.6 respectively. The total percentage value addeded time for resource/maintenance team 1,2 and 3 is 54.54%,53.00% and 40.50% respectively. The total percentage value added time for all the resources is 51.52%. When the two schedules are compared it can be seen that the %value added time for both resource 1 and 2 has increased while resource 3 experienced

Table 5.3: Original maintenance schedule for resource/maintenance team 3

	Mon	**Tue**	**Wed**	**Thur**	**Fri**
Total tasks to be performed	0	4	2	3	4
Total traveling time(hours)	0	4.21	4.04	5.43	5.2
Total servicing time(hours)	0	4.15	2.2	2.95	4.2
Total working time(hours)	0	8.36	6.24	8.38	9.4
% value added time	**0%**	**49.64%**	**35.26%**	**35.20%**	**44.68%**

a slight decrease in %value added time. The total percentage value added time for all the resources has increased from 46.50% to 51.52%.

Table 5.4: Optimised maintenance schedule for resource/maintenance team 1

	Mon	**Tue**	**Wed**	**Thur**	**Fri**
Total tasks to be performed	5	4	5	4	5
Total traveling time(hours)	4.55	3.65	3.39	4.75	4.04
Total servicing time(hours)	4.8	4.95	5.6	4	5.1
Total working time(hours)	9.35	8.6	8.99	8.75	9.14
% value added time	**51.34%**	**57.56%**	**62.30%**	**45.71%**	**55.80%**

Table 5.5: Optimised maintenance schedule for resource/maintenance team 2

	Mon	**Tue**	**Wed**	**Thur**	**Fri**
Total tasks to be performed	4	5	4	4	4
Total traveling time(hours)	4.35	4.37	4.12	3.46	4.68
Total servicing time(hours)	4.6	4.9	4.6	5.05	4.5
Total working time(hours)	8.95	9.27	8.72	8.51	9.18
% value added time	**51.40%**	**52.86%**	**52.75%**	**59.34%**	**49.02%**

Table 5.6: Optimised maintenance schedule for resource/maintenance team 3

	Mon	**Tue**	**Wed**	**Thur**	**Fri**
Total tasks to be performed	1	1	2	1	1
Total traveling time(hours)	2.3	1.8	2.78	2.7	1.3
Total servicing time(hours)	1.2	1.2	2.2	1.5	1.3
Total working time(hours)	3.5	3	4.98	4.2	2.6
% value added time	**34.29%**	**40.00%**	**44.18%**	**35.71%**	**50.00%**

5.2 Sensitivity analysis

Sensitivity analysis forms a critical part of the analysis of results since it gives an indication of how sensitive is the algorithm to changes in input parameters. Sensivity analysis also helps in the identification of critical input parameters. Critical inputs parameters are those parameters that have greater effect on the quality of the solution.

Only one input parameter was analysed, which is the neighborhood size. The neighborhood size represents the number of neighborhood solutions that will be ganerated. The bigger the neighborhood size, the greater are the chances of arriving at a good solution. This is due to the fact that many possible solutions are explored when using a large neighborhood size.

Figure 5.1 shows the graph of the %value added time as a function of the neighborhood size. From the figure it can be seen that a high %value added time value is attained when large neighborhood sizes are explored. The only thing that prevents against the use of large neighborhood sizes is the computational time. The larger the neighborhood size, the greater the computational time required to solve the problem. This results in tradeoff between high computational time and better quality of the solution.

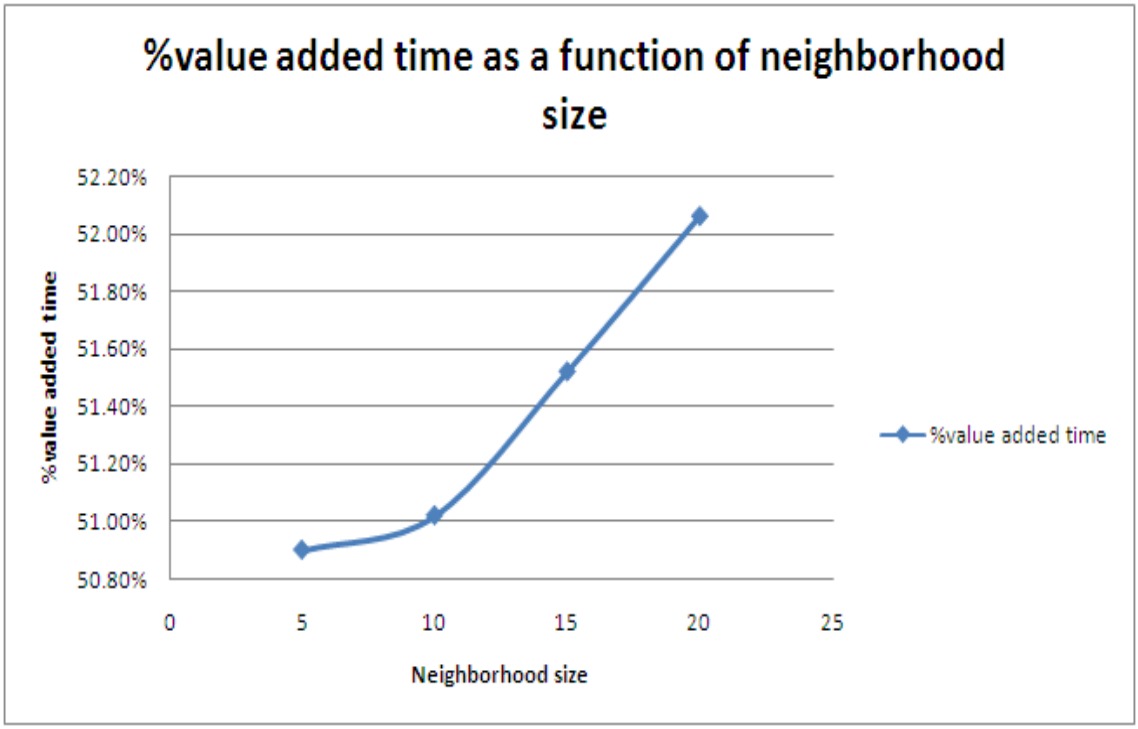


Figure 5.1: Sensitivity of the algorithm to changes in neighborhood size

Chapter 6

Conclusion and future research opportunities

The objective of the project was to optimise maintenance schedules for ABC. It was decided that the MTMCPTD best describes the problem at hand, and the model used to solve the MTMCPTD should be used to solve ABC's problem. It was further discovered that the MTMCPTD is NP-hard and therefore heuristic techniques should be used to solve the problem. A TS heuristic embedded in an AMP was employed to solve the problem. The algorithm was coded in Octave.

An instance of the problem was solved using the algorithm. There was substantial improvement attained with the application of the TS heuristic. Traveling was reduced and the % value added time was increased. It is critical to decrease traveling time while at the same time increase the service time since traveling time is not a value adding activity. Therefore traveling time should be kept as minimal as possible.

This project was only conducted to ascertain if operations research can bring about improvements in the scheduling of maintenance tasks. From the work that was done in this project, it is clear that operations research can bring about improvements in the scheduling of maintenance tasks. A student version octave software was used for the coding of the algorithm. Therefore greater improvements can be made if the algorithm can be coded in a purchased software that will offer more capability. An opportunity for improvement also can be seen in acquiring the services of a qualified programmer to do the coding of the algorithm.

Bibliography

Butt, S. and Cavalier, T. (1994). A heuristic for the multiple tour maximum collection problem. *Computer and Operations Research*, 21:101–111.

Coyle, J., Bardi, E., and Langley, C. (2003). *The management of business logistics*. Thomson South-Western.

Fourier (2010a). http://www.fourier.co.za/fourier

Fourier (2010b). http://www.fourier.co.za/fourier

Goel, A. and Gruhn, V. (2006). *A General Vehicle Routing Problem*. PhD thesis, Chair of Applied Telematics and e-Business, Computer Science Faculty, University of Leipzig, Klostergasse 3, 04109 Leipzig, Germany.

Rajgopal, J. (2004). *Principles and applications of operations research*. In Maynard's industrial engineering handbook. McGraw-Hill.

Rardin, R. (1998). *Optimization in Operations Research*. Prentice Hall.

Solomon, M. (1985). Algorithms for the routing and scheduling problems with time window constraints. *Operations Research*, 35(2):254–265.

Tang, H., Miller, E., and Tomastik, R. (2007). Scheduling technicians for planned maintenance of geographically distributed equipment. *Transportation Research Part E*, 43:591–609.

Vacic, V. and Sobh, T. (2002). *Vehicle Routing with Time Windows*. PhD thesis, Department of Computer Science and Engineering, University of Brigeport, Bridgeport, CT 06604, USA.

Wikipedia (2010). http://en.mwikipedia.org/wiki/facility management.

Winston, W. and Venkataramanan, M. (2003). *Introduction to mathematical programming*. Thomson Brooks/Cole.

Appendix A

The Octave code

```
function [Objective_function,Schedule]=Tabu_search_heuristic(T,m)
tic
%m=the planning horizon
%T=the working hours per day
load traveling_time.txt
load reward.txt
load service_time.txt
No_of_tasks=length(service_time);
p=No_of_tasks;
r=reward;
t=traveling_time;
s=service_time;

%STEP_1
employee=1;
k=1;
for j=1:p-1 %This will record all the tasks that can be performed in one working da
y in a vector
  if (t(1,j+1)+t(j+1,1)+s(j+1))<=T
    unperformed_task(k,1)=j+1;
    k=k+1;
  end
end

%Generate m initial vehicle tours
unscheduled_work=unperformed_task;
while sum(unperformed_task)>0
    [tour_duration,Q,unperformed_task,tour_reward]=initiation(m,t,s,r,T,p,unperform
ed_task);
    advice=[];
    H(1:m,1)=1;
    while (sum(unperformed_task)>0)&(sum(H)>0)
      [tour_duration,Q,unperformed_task,tour_reward]=vertice_insert(m,unperformed_t
ask,t,s,r,T,tour_duration,Q,tour_reward,advice);
      [H,advice]= availability_check(m,unperformed_task,t,s,H,T,tour_duration,Q);
    end
%STEP_2

%Generate initial solution
    initial_solution=Q;
    initial_reward=tour_reward;
    initial_duration=tour_duration;
    Best_solution=initial_solution;
    reward_BS=sum(initial_reward);
    S_f=initial_solution;
    Sf_rew=sum(initial_reward);

%STEP_3 Tabu search

%Set parameters
    alpha=6; %Number of non-improvement iterations
    beta=10;%Neighborhood size
    e=0.05; %Tour duration coefficient
    eta=10; %Penalty parameter
    delta=6; %Retrospect length
    q_tabu=0;%Tabu iterations
    q_m=[];%Retrospect length counter
    tabu_list=[];


%Generate neighborhood solution
  while (q_tabu<alpha)&(sum(unperformed_task)>0)

    [S,Quasi,lists,duration,reward,removed_task,insert_task]=neighborhood(m,p,t,s,r
```

Figure A.1: The Tabu_search_heuristic function (page 1)

```
,T,beta,e,eta,unperformed_task,initial_solution,initial_duration,initial_reward);


%Evaluate the neighborhood solutions
    [eta,q_tabu,q_m,beta,S_f,Best_solution,reward_BS,Sf_rew,initial_solution,initia
l_reward,initial_duration,unperformed_task,tabu_list,unscheduled_work]=evaluate(m,p
,r,t,s,e,T,delta,eta,q_tabu,q_m,alpha,beta,S,Quasi,lists,duration,reward,Best_solut
ion,reward_BS,Sf_rew,S_f,initial_solution,initial_reward,initial_duration,unperform
ed_task,removed_task,insert_task,tabu_list,unscheduled_work);
  end

%STEP_4
%Update the AMP

    Solution(employee,1)={Best_solution};
    Solution_reward(employee,1)=reward_BS
    Feasible_solution(employee,1)={S_f};
    Feasible_solution_reward(employee,1)=Sf_rew;
    employee=employee+1;
end
toc
Solution
Solution_reward
Feasible_solution
Feasible_solution_reward
```

Figure A.2: The Tabu_search_heuristic function (page 2)

```
function [tour_duration,Q,unperformed_task,tour_reward]=initiation(m,t,s,r,T,p,unpe
rformed_task)
Q=[];
Q(1:m,1:3)=1; %Generated tours (pi)
Q(1:m,2)=0;
i=1;
tour_duration=[];
tour_reward=[];
while (i<=m)&(sum(unperformed_task)>0) %Randomly insert vertices into the m tours
  select=ceil(rand*(length(unperformed_task)));
  Q(i,2)=unperformed_task(select,1);
  if select==1
    unperformed_task=unperformed_task(2:(length(unperformed_task)));
  elseif select==(length(unperformed_task))
    unperformed_task=unperformed_task(1:length(unperformed_task)-1);
  else
    unperformed_task=[unperformed_task(1:select-1);unperformed_task(select+1:length
(unperformed_task))];
  end
  i=i+1;
end
for i=1:m % calculate each tours duration and reward
  tour_duration(i,1)=s(Q(i,2),1)+t(1,Q(i,2))+t(Q(i,2),1);
  tour_reward(i,1)=r(Q(i,2),i);
end
```

Figure A.3: The initiation function

```
  function [tour_duration,Q,unperformed_task,tour_reward]=vertice_insert(m,unperfor
med_task,t,s,r,T,tour_duration,Q,tour_reward,advice)
  %Insert as much tasks as possible to the initial solution.
  if isempty(advice)==0
    select=advice(ceil(rand*(length(advice))));
    task=unperformed_task(select,1);
  else
    select= ceil(rand*(length(unperformed_task)));
    task=unperformed_task(select,1);
  end
  task_reward=r(task,1:m);
  counter=0;
  while (counter<1)&((sum(task_reward))>0)
    [max_reward,day]=max(task_reward);
    k=length(find(Q(day,:)));
    time=[];
    for j=1:k-1
      time(j)=t(Q(day,j),task)+t(task,Q(day,j+1))-t(Q(day,j),Q(day,j+1))+s(task);
    end
    [min_time,point]=min(time);
    duration=tour_duration(day,1)+ min_time;
    if duration<=T
      counter=1;
      Q(day,k+1)=1;
      for p=k:-1:point+2
        Q(day,p)=Q(day,p-1);
      end
      Q(day,point+1)=task;
      if select==1
        unperformed_task=unperformed_task(2:(length(unperformed_task)));
      elseif select==(length(unperformed_task))
        unperformed_task=unperformed_task(1:length(unperformed_task)-1);
      else
        unperformed_task=[unperformed_task(1:select-1);unperformed_task(select+1:le
ngth(unperformed_task))];
      end
      tour_duration(day,1)= tour_duration(day,1)+ min_time;
      tour_reward(day,1)= tour_reward(day,1) + r(task,day);
    else
      task_reward(1,day)=0;
    end
  end
```

Figure A.4: The vertice_insert function

```
  function [tour_duration,Q,H,task_list,tour_reward,inserted_task]=vertice_insert_2
(m,task_list,t,s,r,H,T,tour_duration,Q,tour_reward,e,inserted_task,select,recomend)
  % Insert as much tasks as possible to the current neighborhood solution
  if length(recomend)>0
    decide=ceil(rand*(length(recomend)));
    elect=recomend(decide);
  else
    elect= ceil(rand*(length(task_list)));
  end
  task=task_list(elect,1);
  task_reward=r(task,:);
  counter=0;
  while (counter<1)&((sum(task_reward))>0)
    [max_reward,day]=max(task_reward);
    k=length(find(Q(day,:)));
    time=[];
    for j=1:k-1
      time(j)=t(Q(day,j),task)+t(task,Q(day,j+1))-t(Q(day,j),Q(day,j+1))+s(task);
    end
    [min_time,point]=min(time);
    duration=tour_duration(day,1)+ min_time;
    if duration<=(T*(1+e))
      inserted_task(length(inserted_task)+1,1)=task;
      counter=1;
      Q(day,k+1)=1;
      for p=k:-1:point+2
        Q(day,p)=Q(day,p-1);
      end
      Q(day,point+1)=task;
      if (elect==1)&(length(task_list)>1)
        task_list=task_list(2:(length(task_list)));
      elseif (elect==(length(task_list)))&(length(task_list)>1)
        task_list=task_list(1:length(task_list)-1);
      elseif (elect>1)&(elect<length(task_list))
        task_list=[task_list(1:elect-1);task_list(elect+1:length(task_list))];
      else
        task_list=[];
      end
      tour_duration(day,1)= tour_duration(day,1)+ min_time;
      tour_reward(day,1)= tour_reward(day,1) + r(task,day);
    else
      task_reward(1,day)=0;
    end
  end
```

Figure A.5: The vertice_insert_2 function

```
  function [H,advice]= availability_check(m,unperformed_task,t,s,H,T,tour_duration,
Q)
  %Determine whether more tasks can be inserted into the current neighborhood solut
ion
  advice=[];
  for j=1:m
    sum=0;
    k=length(find(Q(j,:)));
    for i=1:length(unperformed_task)
      task=unperformed_task(i);
      time=[];
      for p=1:k-1
        time(p)=t(Q(j,p),task)+t(task,Q(j,p+1))-t(Q(j,p),Q(j,p+1))+s(task);
      end
      if min(time)<=(T-tour_duration(j))
        sum=sum+1;
        advice(length(advice)+1)=i;
      end
    end
    if sum==0
      H(j)=0;
    end
  end
```

Figure A.6: The availability_check function

```
  function [H,recomend]= availability_check_2(m,e,task_list,t,s,H,T,tour_duration,Q
,r)
  %Determine whether more tasks can be inserted into the initial solution solution

  recomend=[];
  for j=1:m
    sum=0;
    k=length(find(Q(j,:)));
    for i=1:length(task_list)
      task=task_list(i);
      time=[];
      for p=1:k-1
        time(p)=t(Q(j,p),task)+t(task,Q(j,p+1))-t(Q(j,p),Q(j,p+1))+s(task);
      end
      reward=r(task,j);
      difference=T*(1+e)-tour_duration(j);
      if (min(time)<=difference)&(reward>0)
        sum=sum+1;
        recomend(length(recomend)+1)=i;
      end
    end
    if sum==0
      H(j)=0;
    end
  end
```

Figure A.7: The availability_check_2 function

```
function [S,Quasi,lists,duration,reward,removed_task,insert_task]=neighborhood(m,p,
t,s,r,T,beta,e,eta,unperformed_task,initial_solution,initial_duration,initial_rewar
d);

S={};
lists={};
Quasi=[];
duration={};
reward={};
insert_task={};
removed_task={};
%Neighborhood generation
%unperformed_task
for iteration=1:beta
  task_list=unperformed_task;
  Q=initial_solution;
  tour_duration=initial_duration;
  tour_reward=initial_reward;
  recomend=[];
  rem_task=[];
  inserted_task=[];
  %Select the two tours to be remove vertices from
  select=[];
  select(1)=ceil(rand*m);
  select(2)=select(1);
  while select(2)==select(1)
    select(2)=ceil(rand*m);
  end
%Determine the current number of vertices in the selected tours
  swap=[];
  choose=[];
  for v=1:2
    k(v)=length(find(Q(select(v),:)));

%Determine the number of vertices to be removed
%Remove two vertices if the tour contains two or more vertices
%Remove one vertice if the tour contains one vertice
    if k(v)==3
      swap(v)=1;
    elseif k(v)>3
      swap(v)=2;
    else
      swap(v)=0;
      while swap(v)==0
        if v==2
          select(2)=select(1);
          while select(2)==select(1)
            select(2)=ceil(rand*m);
          end
        else
          select(1)=select(2);
          while select(1)==select(2)
            select(1)=ceil(rand*m);
          end
        end
        k(v)=length(find(Q(select(v),:)));
        if k(v)==3
          swap(v)=1;
        elseif k(v)>3
          swap(v)=2;
        end
      end
    end
%Randomly choose the vertices to be removed
```

Figure A.8: The neighborhood function (page 1)

```
    for z=1:swap(v)
      if z==2
        choose(v,2)=choose(v,1);
        while choose(v,2)==choose(v,1)
          choose(v,2)=ceil(rand*(k(v)-2))+1;
        end
      else
        choose(v,z)=ceil(rand*(k(v)-2))+1;
      end
    end

%Update the tours(remove the selected vertices)
    if z==2
%Update the reward collected.
      tour_reward(select(v),1)=tour_reward(select(v),1)-r(Q(select(v),choose(v,1)),
select(v))-r(Q(select(v),choose(v,2)),select(v));
%Subtract service time of the removed vertices
      tour_duration(select(v),1)=tour_duration(select(v),1)-s(Q(select(v),choose(v,
1)),1)-s(Q(select(v),choose(v,2)),1);
%Subtract traveling time to and from the removed vertices
      tour_duration(select(v),1)=tour_duration(select(v),1)-t(Q(select(v),choose(v,
1)),Q(select(v),choose(v,1)+1))-t(Q(select(v),choose(v,1)-1),Q(select(v),choose(v,1
)));
      tour_duration(select(v),1)=tour_duration(select(v),1)-t(Q(select(v),choose(v,
2)),Q(select(v),choose(v,2)+1))-t(Q(select(v),choose(v,2)-1),Q(select(v),choose(v,2
)));
      LB=min(choose(v,1),choose(v,2));
      UB=max(choose(v,1),choose(v,2));
      rem_task(length(rem_task)+1,1)=Q(select(v),choose(v,1));
      rem_task(length(rem_task)+1,1)=Q(select(v),choose(v,2));
      if LB==(UB-1)
        tour_duration(select(v),1)=tour_duration(select(v),1)+t(Q(select(v),LB-1),Q
(select(v),UB+1))+t(Q(select(v),LB),Q(select(v),UB));
      else
        tour_duration(select(v),1)=tour_duration(select(v),1)+t(Q(select(v),LB-1),Q
(select(v),LB+1))+t(Q(select(v),UB-1),Q(select(v),UB+1));
      end
      for canc=LB:UB-2
        Q(select(v),canc)=Q(select(v),canc+1);
      end
      for canc=UB-1:k(v)-2
        Q(select(v),canc)=Q(select(v),canc+2);
      end
      Q(select(v),k(v))=0;
      Q(select(v),k(v)-1)=0;
    else
      rem_task(length(rem_task)+1,1)=Q(select(v),choose(v,1));
      Q(select(v),2)=1;
      Q(select(v),3)=0;
      tour_duration(select(v),1)=0;
      tour_reward(select(v),1)=0;
    end
  end
%Insert vertices into the selected tours
  H(1:m,1)=1;
  while (length(task_list)>0)&(sum(H)>0)
    [tour_duration,Q,H,task_list,tour_reward,inserted_task]=vertice_insert_2(m,task
_list,t,s,r,H,T,tour_duration,Q,tour_reward,e,inserted_task,select,recomend);
    [H,recomend]= availability_check_2(m,e,task_list,t,s,H,T,tour_duration,Q,r);
  end
  [S,Quasi,lists,duration,reward,insert_task,removed_task]=store_data(duration,rewa
rd,Quasi,iteration,tour_reward,tour_duration,task_list,S,lists,T,m,eta,Q,inserted_t
ask,insert_task,rem_task,removed_task);
end
```

Figure A.9: The neighborhood function (page 2)

```
function [S,Quasi,lists,duration,reward,insert_task,removed_task]=store_data(durati
on,reward,Quasi,iteration,tour_reward,tour_duration,task_list,S,lists,T,m,eta,Q,ins
erted_task,insert_task,rem_task,removed_task)
  %Calculate the quasi reward for the neighborhood solutions
  Quasi(iteration)=0;
  for i=1:m
    Quasi(iteration)=Quasi(iteration)+tour_reward(i)-(eta*max(tour_duration(i)-T,0)
);
  end

  %Store the duration,reward,inserted tasks,removed tasks and the remaining tasks f
or each neighborhood solution
  duration(iteration,1)={tour_duration};
  reward(iteration,1)={tour_reward};
  lists(iteration,1)={task_list};
  S(iteration,1)={Q};
  insert_task(iteration,1)={inserted_task};
  removed_task(iteration,1)={rem_task};
```

Figure A.10: The store_data function

```
function [max_QR,sol_no,tabu_list]=tabulist(tabu_list,Quasi,removed_task,insert_tas
k,reward_BS)

%Select the best non-tabu solution or use the aspiration criteria for the tabu solu
tion
%The aspiration criteria is that, if the solution is greater than the best solution
 currently then the tabu solution should be  stored as the best solution
  tabu_check=0;
  while (tabu_check==0)&(sum(Quasi)>0)
    [max_QR,sol_no]=max(Quasi);
    check=[];
    insert=[];
    rem_list=removed_task{sol_no,1};
    for i=1:length(rem_list)
      for j=1:rows(tabu_list)
        if rem_list(i)==tabu_list(j,1)
          check(length(check)+1)=tabu_list(j,1);
        end
      end
    end
    if (isempty(check)==1)
      tabu_check=1;
      if isempty(tabu_list)==1
        tabu_list(:,1)=insert_task{sol_no,1};
        tabu_list(:,2)=3;
      else
        tabu_list(:,2)=tabu_list(:,2)-1;
        i=1;
        while i<=rows(tabu_list)
          if (tabu_list(i,2)==0)
            if (i>1)&(i<rows(tabu_list))
              tabu_list=[tabu_list(1:(i-1),:);tabu_list((i+1):rows(tabu_list),:)];
            elseif (i==1)&(rows(tabu_list)>1)
              tabu_list=tabu_list(2:rows(tabu_list),:);
            elseif (i==rows(tabu_list))&(rows(tabu_list)>1)
              tabu_list=tabu_list(1:rows(tabu_list)-1,:);
            else
              tabu_list=[];
            end
          else
            i=i+1;
          end
        end
        insert=insert_task{sol_no,1};
        insert(:,2)=3;
        tabu_list=[tabu_list;insert];
      end
    elseif (isempty(check)==0)&(max_QR>reward_BS)
      tabu_check=1;
      for j=1:length(check)
        i=1;
        while i<=rows(tabu_list)
          if (tabu_list(i,1)==check(j))
            if (i>1)&(i<rows(tabu_list))
              tabu_list=[tabu_list(1:(i-1),:);tabu_list((i+1):rows(tabu_list),:)];
            elseif (i==1)&(rows(tabu_list)>1)
              tabu_list=tabu_list(2:rows(tabu_list),:);
            elseif (i==rows(tabu_list))&(rows(tabu_list)>1)
              tabu_list=tabu_list(1:rows(tabu_list)-1,:);
            else
              tabu_list=[];
            end
          else
            i=i+1;
```

Figure A.11: The tabulist function (page 1)

```
        end
      end
    end


  if isempty(tabu_list)==0
    tabu_list(:,2)=tabu_list(:,2)-1;
    i=1;
    while i<=rows(tabu_list)
      if (tabu_list(i,2)==0)
        if (i>1)&(i<rows(tabu_list))
          tabu_list=[tabu_list(1:(i-1),:);tabu_list((i+1):rows(tabu_list),:)];
        elseif (i==1)&(rows(tabu_list)>1)
          tabu_list=tabu_list(2:rows(tabu_list),:);
        elseif (i==rows(tabu_list))&(rows(tabu_list)>1)
          tabu_list=tabu_list(1:rows(tabu_list)-1,:);
        else
          tabu_list=[];
        end
      else
        i=i+1;
      end
    end
    insert=insert_task{sol_no,1};
    insert(:,2)=3;
    tabu_list=[tabu_list;insert];
  else
    tabu_list(:,1)=insert_task{sol_no,1};
    tabu_list(:,2)=3;
  end

  else
    Quasi(sol_no)=0;
    if sum(Quasi)==0
      max_QR=0;
      sol_no=0;
    end
  end
end
```

Figure A.12: The tabulist function (page 2)

```
function [eta,q_tabu,q_m,beta,S_f,Best_solution,reward_BS,Sf_rew,initial_solution,i
nitial_reward,initial_duration,unperformed_task,tabu_list,unscheduled_work]=evaluat
e(m,p,r,t,s,e,T,delta,eta,q_tabu,q_m,alpha,beta,S,Quasi,lists,duration,reward,Best_
solution,reward_BS,Sf_rew,S_f,initial_solution,initial_reward,initial_duration,unpe
rformed_task,removed_task,insert_task,tabu_list,unscheduled_work)

%Check the tabu status of the generated solutions and update the tabu list.
[max_QR,sol_no,tabu_list]=tabulist(tabu_list,Quasi,removed_task,insert_task,reward_
BS);

%Adjust the cureent parameters according to the tabu status of the generated soluti
ons
adjust=0;
if (max_QR>0)&(sol_no>0)
  initial_solution=S{sol_no,1};
  initial_reward=reward{sol_no,1};
  initial_duration=duration{sol_no,1};
  curent_beta=beta;
  [initial_solution,initial_duration,initial_reward]=improvement(initial_solution,i
nitial_duration,initial_reward,r,t,s,e,T);
  if max_QR>reward_BS
    Best_solution=initial_solution;
    q_tabu=0;
    beta=10;
    reward_BS=max_QR;
    adjust=1;
    if sum(initial_reward)==max_QR
      S_f=Best_solution;
      Sf_rew=sum(initial_reward);
    end

  else
    q_tabu=q_tabu+1;
    if (beta==(10))&(q_tabu>(alpha/2))
      beta=20;
    elseif (beta==(20))&(q_tabu==0)
      beta=10;
    end
  end

 %Adjust the penalty parameter
  len_qm=length(q_m);
  if sum(initial_solution)==max_QR
    q_m(len_qm+1)=1;
  else
    q_m(len_qm+1)=0;
  end

  if len_qm==delta
    sam=sum(q_m);
    if sam>0
      eta=10/2;
    else
      eta=2*10;
    end
    q_m=[];
  end

%Update the vector containing unperformed tasks
  remaining_task=lists{sol_no,1};
  if isempty(remaining_task)==0
    unperformed_task=remaining_task;
  else
    unperformed_task=[];
```

Figure A.13: The evaluate function (page 1)

```
  end

  if (length(unperformed_task)>0)
    rem=removed_task{sol_no,1};
    unperformed_task=[unperformed_task;rem];
  else
    rem=removed_task{sol_no,1};
    unperformed_task=rem;
  end
  if adjust==1
    unscheduled_work=unperformed_task;
  end

else
    if (beta==(10))&(q_tabu>(alpha/2))
      beta=20;
    elseif (beta==(20))&(q_tabu==0)
      beta=10;
    end
    q_tabu=q_tabu+1;
end
```

Figure A.14: The evaluate function (page 2)

```
function [initial_solution,initial_duration,initial_reward]=improvement(initial_sol
ution,initial_duration,initial_reward,r,t,s,e,T)

%VERTEX RELOCATION BETWEEN TWO TOURS

%Select the two tours to compare
select(1)=ceil(rand*(rows(initial_solution)));
select(2)=ceil(rand*(rows(initial_solution)));
while select(1)==select(2)
  select(2)=ceil(rand*(rows(initial_solution)));
end

%Determine the tour with the larger duration
if initial_duration(select(1))>initial_duration(select(2))
  tour=select(1);
  insert=select(2);
else
  tour=select(2);
  insert=select(1);
end

Tot_dura=initial_duration(tour,1)+initial_duration(insert,1);
i=2;
%Perform the relocation if neccessary
while (i<=length(find(initial_solution(tour,:)))-1)
  task=initial_solution(tour,i);
  j=1;
  inserted=0;
  while (j<=length(find(initial_solution(insert,:)))-1)&(inserted==0)
    duration=s(task,1)+t(initial_solution(insert,j),task)+t(task,initial_solution(i
nsert,j+1))-t(initial_solution(insert,j),initial_solution(insert,j+1));
    if duration<=((T*(1+e))-initial_duration(insert,1))
      red_dur=initial_duration(tour,1)-s(task,1)-t(initial_solution(tour,i-1),task)
-t(task,initial_solution(tour,i+1))+t(initial_solution(tour,i-1),initial_solution(t
our,i+1));
      if ((red_dur+duration+initial_duration(insert,1))<=Tot_dura)&(r(task,insert)>
0)
        inserted=1;
        for k=i:length(find(initial_solution(tour,:)))-1
          initial_solution(tour,k)=initial_solution(tour,k+1);
        end
        initial_solution(tour,k+1)=0;
        q=length(find(initial_solution(insert,:)));
        initial_solution(insert,q+1)=1;
        for p=q:-1:j+2
          initial_solution(insert,p)=initial_solution(insert,p-1);
        end
        initial_solution(insert,j+1)=task;
        initial_reward(tour,1)=initial_reward(tour,1)-r(task,tour);
        initial_reward(insert,1)=initial_reward(insert,1)+r(task,insert);
        initial_duration(tour,1)=red_dur;
        initial_duration(insert,1)=initial_duration(insert,1)+duration;
        i=i-1;
      end
    end
    j=j+1;
  end
  i=i+1;
end
```

Figure A.15: The improvement function

Appendix B

The reward matrix

Reward Matrix

	Day				
Task	Monday	Tuesday	Wednesday	Thursday	Friday
1	0	0	0	0	0
2	3	3	3	5	8
3	3	3	5	8	0
4	5	8	0	0	0
5	3	3	3	5	8
6	3	3	5	8	0
7	3	3	3	5	8
8	3	5	8	0	0
9	8	0	0	0	0
10	5	8	0	0	0
11	3	5	8	0	0
12	3	3	5	8	0
13	5	8	0	0	0
14	3	3	3	5	8
15	3	3	5	8	0
16	3	5	8	0	0
17	8	0	0	0	0
18	5	8	0	0	0
19	3	3	5	8	0
20	3	3	3	5	8
21	5	5	8	0	0
22	3	3	3	5	8
23	3	3	5	8	0
24	5	8	0	0	0
25	3	3	3	5	8
26	3	3	5	8	0
27	3	3	3	5	8
28	3	5	8	0	0
29	8	0	0	0	0
30	5	8	0	0	0
31	3	5	8	0	0
32	3	3	5	8	0
33	5	8	0	0	0
34	3	3	3	5	8
35	3	3	5	8	0
36	3	5	8	0	0
37	8	0	0	0	0
38	5	8	0	0	0
39	3	3	5	8	0
40	3	3	3	5	8
41	8	0	0	0	0
42	3	3	5	8	0
43	5	8	0	0	0
44	5	8	0	0	0
45	3	5	8	0	0
46	3	3	3	5	8
47	3	5	8	0	0
48	3	3	3	5	8
49	5	8	0	0	0
50	3	5	8	0	0
51	8	0	0	0	0

Figure B.1: The reward matrix

Appendix C

The service time vector

Service Time

Task No:	Task duration(hours)
1	0
2	1.4
3	1.5
4	1
5	1.3
6	1.5
7	1
8	1.2
9	1
10	1.1
11	1.3
12	1.5
13	1
14	0.5
15	0.8
16	1.4
17	1.3
18	1.4
19	1
20	1.3
21	1
22	1.2
23	1.3
24	1.3
25	1.2
26	1
27	1
28	1.2
29	1.1
30	1
31	1
32	1.2
33	1.3
34	1.2
35	1
36	1
37	1.2
38	1.4
39	1.2
40	1.2
41	0.5
42	0.75
43	1.25
44	1
45	1
46	0.8
47	1.2
48	1
49	0.5
50	1
51	1

Figure C.1: The service time vector

Appendix D

The traveling time matrix

Traveling Time

	Time(hours)																									
Task No:	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26
1	0	0.65	0.6	0.85	0.95	1.35	1	0.85	0.85	1.25	0.55	0.95	0.6	0.9	0.65	0.85	0.65	0.95	1.15	1.15	0.85	0.6	1.1	1.05	1.1	0.95
2	0.65	0	1.4	1.05	0.7	1.5	0.95	1.5	1.1	0.75	1.3	1.05	0.9	0.7	1.4	0.6	1.3	0.7	0.85	0.75	1.05	0.6	0.95	0.55	1.25	0.85
3	0.6	1.4	0	0.6	1.4	1.2	1.15	1	0.65	1	1.4	0.6	1.2	0.7	0.8	0.95	1.25	0.7	1.1	0.95	1.4	0.55	1.15	1	0.75	1.45
4	0.85	1.05	0.6	0	0.85	0.6	1.15	1.25	1.2	1.1	0.55	1.5	0.65	0.65	1.2	0.75	1.1	0.7	0.6	1.05	1.4	0.95	0.9	1.4	0.7	1
5	0.95	0.7	1.4	0.85	0	0.9	1.05	0.75	0.75	1.4	0.85	0.95	1.2	0.7	1.15	1.15	0.6	0.7	1.05	1.25	1.2	0.8	1.2	0.95	0.8	0.55
6	1.35	1.5	1.2	0.6	0.9	0	0.55	1.5	1.4	1.1	0.65	0.85	0.55	1.2	1.4	1.2	1.35	1.2	0.65	0.75	0.55	0.55	1.2	0.7	0.75	0.9
7	1	0.95	1.15	1.15	1.05	0.55	0	1.35	1.2	1.05	1.5	0.9	1.15	1	1.3	1.2	1.3	1	0.6	1.1	1.25	0.9	1.15	1.1	1.5	1
8	0.85	1.5	1	1.25	0.75	1.5	1.35	0	1.25	1.05	1.05	1.1	0.95	1.1	0.85	1.45	0.95	1.35	1.1	0.7	0.55	1.3	0.85	1.05	1.15	0.6
9	0.85	1.1	0.65	1.2	0.75	1.4	1.2	1.25	0	0.6	1.25	1.1	1.1	1.05	1	0.6	1.35	0.65	1.4	1.4	1.1	0.65	1	0.6	1.2	1.05
10	1.25	0.75	1	1.1	1.4	1.1	1.05	1.05	0.6	0	1.1	1.5	1.3	0.6	1.4	1.3	1.05	0.65	1.35	0.95	0.75	0.8	1.1	0.85	1.05	0.75
11	0.55	1.3	1.4	0.55	0.85	0.65	1.5	1.05	1.25	1.1	0	1.15	0.8	0.65	1.05	0.85	1.4	0.55	0.9	1.1	0.65	0.9	1.1	0.75	0.75	0.7
12	0.95	1.05	0.6	1.5	0.95	0.85	0.9	1.1	1.1	1.5	1.15	0	0.9	1.4	0.65	0.6	0.8	1.3	1.45	1.1	1.15	1.3	0.55	1.05	1.35	1.15
13	0.6	0.9	1.2	0.65	1.2	0.55	1.15	0.95	1.1	1.3	0.8	0.9	0	0.85	1.25	1.1	0.85	0.55	0.7	0.8	0.9	0.65	1	1.5	1.3	0.7
14	0.9	0.7	0.7	0.65	0.7	1.2	1	1.1	1.05	0.6	0.65	1.4	0.85	0	1	0.95	1.05	1.4	1.45	0.85	1.25	1.35	1.3	0.9	0.8	1.3
15	0.65	1.4	0.8	1.2	1.15	1.4	1.3	0.85	1	1.4	1.05	0.65	1.25	1	0	1.25	1	0.8	1	1.5	1.15	1.35	0.65	1.1	0.75	1.35
16	0.85	0.6	0.95	0.75	1.15	1.2	1.2	1.45	0.6	1.3	0.85	0.6	1.1	0.95	1.25	0	0.75	0.6	1.3	1.3	1	0.85	1.35	1.35	1.35	0.95
17	0.65	1.3	1.25	1.1	0.6	1.35	1.3	0.95	1.35	1.05	1.4	0.8	0.85	1.05	1	0.75	0	0.85	1.5	1.4	1.1	0.75	1.5	0.7	1.3	0.6
18	0.95	0.7	0.7	0.7	0.7	1.2	1	1.35	0.65	0.65	0.55	1.3	0.55	1.4	0.8	0.6	0.85	0	0.75	0.75	1.05	0.9	1.15	0.65	1.4	1.4
19	1.15	0.85	1.1	0.6	1.05	0.65	0.6	1.1	1.4	1.35	0.9	1.45	0.7	1.45	1	1.3	1.5	0.75	0	0.55	0.85	1.4	1.35	1.45	0.8	1.25
20	1.15	0.75	0.95	1.05	1.25	0.75	1.1	0.7	1.4	0.95	1.1	1.1	0.8	0.85	1.5	1.3	1.4	0.75	0.55	0	0.6	0.7	0.95	1.45	1.2	0.8
21	0.85	1.05	1.4	1.4	1.2	0.55	1.25	0.55	1.1	0.75	0.65	1.15	0.9	1.25	1.15	1	1.1	1.05	0.85	0.6	0	1.3	1.4	1.35	1.4	1.1
22	0.6	0.6	0.55	0.95	0.8	0.55	0.9	1.3	0.65	0.8	0.9	1.3	0.65	1.35	1.35	0.85	0.75	0.9	1.4	0.7	1.3	0	1.25	0.7	1.05	1.5
23	1.1	0.95	1.15	0.9	1.2	1.2	1.15	0.85	1	1.1	1.1	0.55	1	1.3	0.65	1.35	1.5	1.15	1.35	0.95	1.4	1.25	0	0.95	1	1.1
24	1.05	0.55	1	1.4	0.95	0.7	1.1	1.05	0.6	0.85	0.75	1.05	1.5	0.9	1.1	1.35	0.7	0.65	1.45	1.45	1.35	0.7	0.95	0	0.6	1.2
25	1.1	1.25	0.75	0.7	0.8	0.75	1.5	1.15	1.2	1.05	0.75	1.35	1.3	0.8	0.75	1.35	1.3	1.4	0.8	1.2	1.4	1.05	1	0.6	0	0.8
26	0.95	0.85	1.45	1	0.55	0.9	1	0.6	1.05	0.75	0.7	1.15	0.7	1.3	1.35	0.95	0.6	1.4	1.25	0.8	1.1	1.5	1.1	1.2	0.8	0
27	0.6	1.4	0.55	1.5	0.7	0.85	1.4	1.1	0.6	0.7	1.2	0.65	1.25	1.3	1.2	0.95	0.75	0.75	1.25	0.55	0.85	1.25	0.6	0.9	1.15	1.1
28	0.85	0.95	1.35	1.35	1	1.1	1.3	1.35	1.5	0.55	1.25	0.85	0.6	1.2	0.85	0.95	0.7	0.85	1.35	0.8	1.3	1.4	0.6	1.3	1.3	0.85
29	1.5	1.45	1.4	1	0.9	1.1	0.9	1.25	0.8	0.6	1	1.3	0.7	0.6	0.95	0.65	1.25	0.75	0.55	1.4	1.25	1.45	1.3	1.35	0.75	1.35
30	0.6	0.85	1	0.55	0.95	1.45	1.45	0.55	0.6	0.8	0.95	0.6	1.4	0.6	0.6	1.25	1.5	1.5	0.95	0.85	1.4	0.6	1.45	0.7	1.45	0.9
31	0.65	1.35	1.25	1.05	0.6	1.5	0.65	1.2	0.65	1.4	1.1	0.6	1.35	1	1.15	0.95	1.35	0.95	1.25	1.1	1.4	1.25	0.55	0.8	1.05	0.75
32	0.9	0.55	1.3	0.6	0.7	1.15	0.75	1	1.15	1.5	1.15	1.5	1	0.7	1.45	1.15	0.65	1	0.85	0.55	0.6	1.45	1.05	0.7	0.75	0.55
33	1.1	1.35	1.3	0.95	0.85	1	0.7	0.95	1.05	0.6	1.1	0.85	0.75	0.8	0.6	0.65	1.5	0.6	0.95	1.05	1.3	1.1	1.25	1.25	0.8	0.95
34	1.15	1.5	0.8	1.05	1.35	0.95	0.7	1	1.15	0.95	1.4	1.1	0.7	0.8	0.75	1.15	1.4	1.25	1.2	1.45	1.05	0.7	1.25	0.85	1.45	0.85
35	1.35	0.65	1.45	0.85	1.2	0.65	1	1.5	1.25	0.75	1	0.7	1.3	1.05	1.25	0.7	0.95	1.1	1.2	0.65	0.95	1.15	1.5	0.6	0.65	0.65
36	1.3	0.65	0.75	1.45	0.55	0.8	1.45	1.45	0.85	0.7	0.95	0.6	1.5	1.3	1.15	0.9	1.35	1.4	1	1.2	0.65	0.65	0.7	0.6	0.65	1.05
37	1.05	1.4	0.75	1	1.35	1.15	1.45	1.1	1.15	0.8	0.9	1.05	0.6	1.3	1.45	1.5	0.7	0.85	1.15	1.1	1.25	1.2	1.25	1	0.7	1.25
38	0.85	1.4	1.3	1.1	0.9	1.25	1.15	0.6	1.05	1.1	0.55	0.65	0.85	1.35	0.95	1.5	0.75	1	1.35	1.3	1.15	1.35	1.25	1.1	0.85	1.45
39	1.25	1.45	1.25	0.95	0.6	0.95	1.05	1.5	0.7	1	1.15	1.5	0.75	1.3	1	1.35	1.25	0.65	0.9	0.8	1.15	0.6	0.95	1.25	1.35	1.3
40	1.45	0.9	0.7	1.15	0.75	1.45	1.05	1.45	0.95	1.4	0.9	1.1	1.45	0.85	1	1.5	0.7	0.85	0.95	0.85	1.4	1.05	0.75	1.05	0.85	1
41	1.38	1.02	1.41	0.91	0.97	0.99	1.07	0.83	0.6	0.83	0.6	1.09	1.22	1.15	1.5	0.66	1.02	0.57	0.7	1.2	0.81	1.3	1.14	0.58	0.9	1.08
42	1.24	1.23	0.52	0.84	0.93	0.99	0.92	1.45	1.29	1.05	0.83	0.69	1.26	1.22	1.03	0.76	0.92	0.85	1	1.14	0.71	0.53	1.01	1.49	0.81	1.07
43	0.71	0.85	1.5	0.51	0.95	0.8	1.17	1	1.19	0.86	1.31	0.58	0.99	0.61	0.84	0.83	0.82	0.61	1.24	0.79	0.69	0.52	0.54	1.22	0.66	0.74
44	0.9	1.3	0.9	1.35	0.77	1.26	0.74	0.52	1.33	0.62	0.94	1.04	0.75	1.16	0.54	1.49	1.06	1.18	1.43	1.07	1.34	0.95	0.51	0.56	1.29	1.36
45	0.81	0.93	1.22	0.69	1	0.99	1.3	0.91	0.74	1.5	0.75	0.65	0.99	0.75	1.48	1.42	1.02	0.51	1.24	1.03	0.9	1.2	1.49	0.8	1.24	0.52
46	1.44	0.88	1.3	0.6	1.5	1.03	1.37	0.54	1.45	1.15	1.18	1.12	0.92	0.79	1.43	1.22	1.45	0.89	0.99	0.95	0.74	0.69	1.33	1.04	1.13	1.34
47	1.16	1.48	1.21	0.62	0.74	1.37	0.62	1.42	0.75	1.49	1.43	0.99	0.86	1.36	1.35	1	1.36	1.32	1.21	0.67	0.85	0.66	0.51	0.99	1.45	0.72
48	1.17	1	0.9	0.53	0.91	0.95	1.06	0.71	1.14	1.17	0.83	1.12	1.24	1.01	0.74	1.41	0.8	1.08	0.51	0.53	0.58	1.15	0.89	0.62	0.77	0.8
49	0.78	1.01	1.39	1.2	0.86	0.99	0.77	0.85	0.63	1	1.2	0.63	1.38	1.18	0.8	1.35	0.86	1.03	1.23	0.69	1.42	1.09	1.45	1.38	1.4	0.66
50	1.39	0.8	0.67	1.22	1.27	1.5	0.99	1.03	1.42	1	0.51	0.79	0.81	1.05	0.89	0.59	0.68	0.71	1.12	1.15	1.22	0.56	1.06	1.23	1.44	1.11
51	0.53	1.36	1.41	1.09	1.11	1.36	0.69	1.03	0.8	1.37	1	0.9	1.35	0.82	1.46	0.63	1.19	0.99	1.22	1.06	0.9	0.72	0.65	1.17	1.14	0.68

Figure D.1: The traveling time matrix (page 1)

| Task No. | Time(hours) | | |

Appendix E

Original schedule

Weekly Schedule(Before optimisation)

	Day				
Tasks:	Monday	Tuesday	Wednesday	Thursday	Friday
1					
2					1
3				1	
4		1			
5					1
6				1	
7					1
8			1		
9	1				
10		1			
11			1		
12				1	
13		1			
14					1
15				1	
16			1		
17	1				
18		1			
19				1	
20					1
21			1		
22					1
23				1	
24		1			
25					1
26				1	
27					1
28			1		
29	1				
30		1			
31			1		
32				1	
33		1			
34					1
35				1	
36			1		
37	1				
38		1			
39				1	
40					1
41	1				
42				1	
43		1			
44		1			
45			1		
46					1
47			1		
48					1
49		1			
50			1		
51	1				

Figure E.1: The original schedule

Maintenance team 1:

1	9	17	29	1	0
1	4	10	13	18	1
1	8	11	16	21	1
1	3	6	12	15	1
1	2	5	7	14	1

Maintenance team 2:

1	37	41	51	1	0
1	24	30	33	1	0
1	28	31	36	45	1
1	19	23	26	32	1
1	20	22	25	27	1

Maintenance team 3:

1	1	0	0	0	0
1	38	43	44	49	1
1	47	50	1	0	0
1	35	39	42	1	0
1	34	40	46	48	1

Figure E.2: The original schedule for workers

Appendix F

Optimised schedule

Weekly Schedule(After optimisation)

	Day				
Tasks:	Monday	Tuesday	Wednesday	Thursday	Friday
1					
2					1
3				1	
4	1				
5			1		
6				1	
7			1		
8			1		
9	1				
10		1			
11			1		
12				1	
13		1			
14					1
15				1	
16			1		
17					1
18		1			
19		1			
20					1
21			1		
22					1
23				1	
24	1				
25					1
26				1	
27					1
28		1			
29	1				
30		1			
31			1		
32		1			
33	1				
34	1				
35				1	
36			1		
37	1				
38		1			
39				1	
40					1
41	1				
42				1	
43		1			
44	1				
45			1		
46					1
47			1		
48					1
49		1			
50			1		
51	1				

Figure F.1: The optimised schedule

Maintenance team 1:

1	51	4	41	33	44	1
1	18	10	28	43	1	0
1	11	50	36	5	31	1
1	15	39	35	26	1	0
1	22	2	14	48	27	1

Maintenance team 2:

1	24	9	29	37	1	0
1	13	38	19	30	49	1
1	16	21	8	45	1	0
1	23	12	42	3	1	0
1	20	46	40	25	1	0

Maintenance team 3:

1	34	1	0
1	32	1	0
1	47	7	1
1	6	1	0
1	17	1	0

Figure F.2: The optimised schedule for workers