On-Demand Air Transportation Flight Scheduling.

Tanya Lerlin La Foy

School of Computational and Applied Mathematics,
University of the Witwatersrand,
Johannesburg, South Africa.

A dissertation submitted for the degree of
Master of Science.

May 7, 2012

Declaration

I declare that this dissertation is my own, unaided work. It is being submitted as partial fulfillment for the Degree of Master of Science to the University of the Witwatersrand, Johannesburg. It has not been submitted before for any degree or examination in any other University.

_____

May 7, 2012

Abstract

On-demand air transportation is a recent trend in the airline industry. It allows the customer to call in days or even hours before to book a flight. Therefore, the scheduling and planning of this type of airline needs to be done daily. Hence, a successful on-demand air transportation requires an efficient flight scheduling system to construct the optimal daily flight schedules. An on-demand air transportation flight scheduling problem that arose in a Southern African industry has been studied. A new solution methodology is proposed. A number of new heuristics are used to combine flight legs for a robust solution. A time-space multi-commodity network is introduced to derive the mathematical model which is then solved using CPLEX. The results obtained are then compared with known results showing much more efficient performances and saving for the industry.

Acknowledgements

First and foremost, I give thanks to God for his blessings and guidance in my life.

I am indebted to many individuals who have helped me during the preparation of this dissertation. I would like to express my sincere appreciation to my supervisor, Prof M. Ali, for his tireless guidance, patience and encouragement in doing this research. Without his supervision this work could not have been possible. I am confident that my future employers will benefit from the skills he has helped me develop.

My sincere thanks also goes to Neil Lumsden, the managing director of Sefofane. I have had many discussions with him during the initial period of this research. Neil Lumsden also gave us all the necessary data needed for this research.

I am particularly thankful to Ian Campbell and Mark Silverwood, School of Mechanical, Aeronautical and Industrial Engineering, for their help and guidance. Ian Campbell assisted me to get the problem from Sefofane, a local air transportation company. I have also had numerous discussions with Ian Campbell during the course of this research. I am also grateful to Mark Silverwood for his help during the initial phase of this research. He took great patience to explain the scheduling problem to me. In particular, I thankfully acknowledge his full support for the Matlab program needed in this research, and for the heuristic suggested in the thesis .

I want to thank my family for their love, support and devotion. Their encouragement has given me the strength and motivation to complete this dissertation. Many thanks go to my grandmother - Sally La Foy, my mother - Charnel La Foy, my stepfather - Wilbur Kemp, my sisters - Tasmin Kemp and Tiffany Kemp, my aunts and my uncles for their constant inspiration.

Special thanks goes out to Clinton Solomons and Terry-Leigh Oliphant for their constant support, communication and love.

I appreciate the financial support from the NRF.

Lastly, my gratitude goes out to the School of Computational and Applied Mathematics.

Contents

1 Introduction **1**

1.1 Background 1

1.2 Solution methodologies 4

1.3 On-demand air transportation 4

1.4 Sefofane and its flight scheduling problem 6

1.5 Organization of the dissertation 8

2 Problem description **10**

2.1 Components of Sefofane's flight scheduling and operating factors . . . 13

2.1.1 The booking list 14

2.1.2 Passenger type 15

2.1.3 The earliest and latest departure and arrival times 16

2.1.4 The maximum number of intermediate stops 16

2.1.5 Freight 17

2.1.6 Aircraft type, maintenance and refueling 17

2.1.7 The pilots 18

2.2 The manual schedule at Sefofane 19

2.3 A sample problem 20

2.3.1 Flight time and cost calculation 24

2.4 Effects of optimization in flight scheduling 26

2.5 The Sefofane scheduling problem 30

3 The multi-commodity network flow formulation 34

3.1 The flow network 34

3.1.1 The flow network structure 35

3.1.2 The aircraft fleet network structure 37

3.2 The mathematical model using the multi - commodity network flow . 42

3.2.1 The mathematical model for the basic fleet assignment problem 43

3.2.2 The fleet assignment model with time windows 47

4 Fleet and flight leg assignment with time windows 53

4.1 Heuristics to create direct and indirect flight legs 54

4.2 The creation of the aircraft fleet network 60

4.2.1 Direct and indirect flight legs 61

4.2.2 The trip time of the direct and indirect flight legs 64

4.2.3 The departure and arrival time windows 65

4.2.4 The aircraft fleet network 67

4.3 The mathematical model 69

5 Computational results 75

5.1 Equipment and software specifications 75

5.2 Manual schedule vs model schedule 81

6 Conclusion 83

Appendix I 85

Appendix II 88

Appendix III 88

Appendix IV 91

Appendix V 92

Appendix VI **94**

Appendix VII **96**

Appendix VIII **98**

Appendix IX **101**

Appendix X **104**

Appendix XI **107**

Appendix XII **117**

Appendix XIII **124**

Bibliography **181**

List of Figures

2.1 A map of the Okavango Delta 11
2.2 Psuedo-network of the solution in Table 2.5 23
2.3 Psuedo-network of the solution of Scenario 1 28
2.4 Psuedo-network of the solution of Scenario 2 29
2.5 Psuedo-network of the solution of Scenario 3 30

3.1 A flow network 35
3.2 A flow network 37
3.3 A time space aircraft network 38
3.4 The active flight arcs and ground arcs in the time interval [6:10, 6:20] 46
3.5 Example of the weakness of the basic fleet assignment model 47
3.6 Example of a given original flight leg 49
3.7 Example flight legs with time windows 49

4.1 Creation of an indirect flight leg using ***Indirect*** 58
4.2 Creation of extended indirect flight leg using ***Extension*** 60
4.3 Creation of indirect flight legs using ***Indirect*** 63
4.4 Creation of extended indirect flight leg N12 using ***Extension*** 63
4.5 The aircraft flow network for the flight legs in Table 4.11 69

List of Tables

2.1 A daily reservation list 15
2.2 Summary of the booking list Table 2.1 15
2.3 Set of available aircraft and their specifications 21
2.4 Set of available pilots and their specifications 22
2.5 A schedule for the sample problem 31
2.6 Cost of each flight in Table 2.5 31
2.7 Flight schedule for Scenario 1 32
2.8 Flights schedule for Scenario 2 33
2.9 Flight schedule for Scenario 3 33

4.1 The subset of a daily booking list for ***Direct*** 55
4.2 Resultant direct flight leg using ***Direct*** 56
4.3 The subset of a daily booking list where ***Direct*** fails 56
4.4 The subset of a daily booking list for ***Indirect*** 57
4.5 Resultant indirect flight legs using ***Indirect*** 58
4.6 The subset of a daily booking list 61
4.7 The direct flight legs created using ***Direct*** 61
4.8 The flight bookings in Table 4.6 that could not be joined by ***Direct*** . 62
4.9 The indirect flight legs created using ***Indirect*** 62
4.10 The extended indirect flight leg created using ***Extension*** 63
4.11 Additional information of the flight legs in Tables 4.7 - 4.10 66

4.12 Record of flight arcs 70

5.1 Computer specifications 76
5.2 The summarised results for the sets of flight bookings used 80
5.3 Manual schedule vs our schedule 82

Chapter 1

Introduction

1.1 Background

Airline scheduling and planning is one of the most important problems in the transportation industry. Airlines operate in a highly competitive environment, which increases their attempt to secure customers. However, matching the customers expectations does not ensure the maximization of profit, since the airline's schedule planning represents the most critical service with regards to its revenue [24]. Airlines are now reducing cost through better use of resources and improving their scheduling decisions to remain competitive. This is because a 'bad' flight schedule could adversely affect an airline's revenue causing losses [24]. Therefore, the *airline scheduling problem* has been a primary focus of airlines and it has a bearing on its profitability, its level of service and its competitiveness in the market [1].

The *airline scheduling problem* has been of concern to the operations research community for the past 30 years [24]. This is due to the complexity and large magnitude of the *airline scheduling problem*. Operations research has had a positive impact on the planning of the airlines' operations. Applications of operations research to the *airline scheduling problem* goes back to 1994 when Delta airline [32] successfully implemented a fleet assignment problem which saved them millions of

dollars.

The *airline scheduling problem* has been divided into four main sub-problems, namely:

Schedule generation

The schedule generation problem is divided into two stages:

Route development

Route development is the problem in which an airline decides which location pairs it wants to provide as a route, primarily based on forecasted demand information.

Schedule design

This is the most complicated sub-problem and is traditionally divided into two sequential phases such as:

(1) *frequency planning* - planners match the frequency of each flight according to the forecasted demand.

(2) *timetable development* - planners need to decide at which time slots the flights need to be offered. The final results of this is a list of flight legs[1].

In the literature, there are a limited number of optimization models in the area of schedule generation [29]. Schedule generation involves strategic decisions of an airline and its competitors, which is hard to capture in a mathematical model. Teodorovic and Krcmar-Novic [5] presented a methodology that determines optimal flight frequencies. Berge [28] also presented a sub-timetable optimization in schedule design. However, recent models in this area focus on improving an existing schedule. For example, Marten et al. [31] presented a framework for incremental schedule design. Rexing et al. [30] presented a model to improve the schedule generation that allows the flight times to vary within a given time window.

Fleet assignment

[1] A flight from a specific departure location to an arrival location, at a given departure and arrival time.

The fleet[2] assignment sub-problem assigns aircraft fleet types, each having different capacities, to the selected flight legs [15]. These fleet types are assigned based on their equipment capabilities and availabilities, operational costs, and potential revenues. The driving force for this assignment is that different fleet types produce different revenues when assigned to specific flight legs because of their seating capacity and operating costs [15]. Hence, the fleet assignment problem impacts an airline's operations.

Most traditional approaches solve the fleet assignment problem in isolation using the sequential method i.e. the fleet assignment problem is not solved as an integral part of the *airline scheduling problem*. This is normally done with the use of a mixed integer multi-commodity network flow model [23], which depends on an airline's flight network [15]. The time-space network has largely become the most frequent choice of flight network construct [15]. A time-space network uses 'nodes' to represent airports at specific times, and 'arcs' to represent flight legs.

Aircraft and Maintenance routing

This sub-problem assigns each flight leg to a specific individual aircraft within a fleet such that the maintenance requirements for each aircraft are met. This ensures that all the flight legs in the flight schedule respect the aircraft's maintenance conditions.

The maintenance routing problem is jointly addressed with the fleet assignment problem. The inputs of this sub-problem is the output of the fleet assignment problem, otherwise known as a fleeted flight schedule, and the available number of aircraft in each fleet type [24, 27].

Crew scheduling or crew pairing

Crew scheduling entails allocating crew to specific aircraft for specific flight legs, in order to minimize crew costs. There are many constraints to consider for this sub-problem, such as 'governmental and contractual regulations', airline specific rules and labor agreements [17, 18, 22]. The crew scheduling problem is usually

[2]Fleet refers to the different types of aircraft.

formulated as a set partitioning problem [16].

1.2 Solution methodologies

Here we review the solution methodologies used to solve the *airline scheduling problem.*

The advantages of integrating all the sub-problems and solving the entire system is outweighed by it's disadvantages of slow run times and complexity. Therefore, the *airline scheduling problem* is solved sequentially such that the output of the one sub-problem is the input of the subsequent sub-problem. However, this approach has also decreased the accuracy of the solution. The main drawback of this approach is that optimal solution for one sub-problem may not necessarily be optimal for the entire system. Furthermore, the solution of one sub-problem may not be a feasible input for the subsequent sub-problem [2]. Hence this approach results in sub-optimal solutions.

The interdependence of the stages of the sequential approach has motivated researchers to focus on semi-integrated models, which solves two or more sub-problems simultaneously. These have proven to result in significant savings [19, 20]. For example, Lohantepanont and Barnhart [19] presented an integrated model that the optimizes the schedule design and the assignment of aircraft fleet types by observing the demand and supply interaction. Barnhart et al. [20] also introduced a semi-integrated model for the fleet assignment and the aircraft routing problem. The solution approach adopted in this thesis is also based on a semi-integrated model.

1.3 On-demand air transportation

The *airline scheduling problem* has become more complex due to the nature of the demands. Apart from the commercial airlines, many small scale airlines, often

regional-based, have emerged throughout the world. These are well known as on-demand air transportation services. The scheduling and planning of these small scale airlines needs to be carried out on a daily basis. On-demand air transportation allows customers to call in days or even hours before their flight to make a booking request. Their booking request consists of specific departure and arrival locations, as well as specific departure and arrival times. Unlike commercial airlines, an on-demand air transportation service offers a unique scheduling problem due to its highly variable nature, this often requires that the flight schedule be completed daily. Therefore, apart from the computational effects there are complex demand constraints that need to be considered.

All the sub-problems discussed earlier are amongst the several most important problems that arise in any airline industry, and are very interesting problems for the application of optimization techniques. These problems are widely known in the context of commercial airlines, but also occur in on-demand air transportation. The problem this research will be focusing on is on-demand air transportation. On-demand air transportation has the following key features:

- Point to point travel - generally using small airports/airstrips.
- Unplanned - customers set their flight schedules.
- Small aircraft - seating one to twelve passengers.
- Flight times - usually a maximum of one hour.

There are various forms of on-demand air transportation. For example, an on-demand air transportation company does not only schedule the aircraft to meet the customers' demands but also rents out aircraft. This enables any person to call in and hire an aircraft for their own personal request. The request includes: where customers would like to be picked up; where they would like to be transported to; and a specific departure and arrival time. This type of on-demand air transportation is often referred to as air chartering.

In addition to the above consideration, an on-demand airline company may have a fractional ownership business. The fractional ownership concept allows a person to purchase shares of an aircraft. The amount of shares bought by the person depends on the amount of hours the person wants to fly per year. This fractional owner can then use the aircraft to satisfy any of his/her requests within the allowed time boundaries.

A few models were introduced to solve instances of on-demand air transportation scheduling problem [8, 11, 3, 7]. A decision support system for scheduling charter aircraft which assigns a set of demanded flights to the available aircraft at minimal cost was presented by Ronen [8]. An optimization system for Bombardier Flexjet [25] fractional aircraft ownership business representing the aircraft itineraries and crew schedules was developed by Hicks et al. [11]. In addition, Keskinocak and Tayur [7] studied the fractional aircraft scheduling problem for a single type of aircraft with no crew duty restrictions. Martin et al. [9, 10] extended the method developed by Keskinocak and Tayur [7] with multiple types of aircraft and crew constraints.

The *airline scheduling problem* we study here arose in a local taxi-service type on-demand air transportation company. We present a brief overview of this flight scheduling problem in the next section.

1.4 Sefofane and its flight scheduling problem

The research carried out in this thesis arose in Sefofane [26]. Sefofane is a charter airline based in Botswana, Maun[3]. It has 13 passenger aircraft consisting of three different fleet types, which operate in response to tourist demands. These aircraft fly tourists, which arrive at Maun's international airport, to and from safari camps situated in the Okavango Delta[4]. The tourists' daily demands must be satisfied making Sefofane different from many charter airlines where full demand satisfaction

[3]Maun is a town in Botswana, and is one of Sefofane's main hubs.

[4]The details of the Okavango Delta will be explained in the next chapter.

may not be required.

Sefofane normally makes use of small light aircraft, namely the Cessna, and others such as the Pilatus and the King Air. These aircraft typically seat 1 to 12 passengers and only require a single pilot to fly them, but an aircraft type such as Cessna requires a co-pilot. Sefofane's aircraft operate in remote locations which only have small dirt airstrips. These airstrips are over short distances within 500 kilometers (km).

Sefofane experiences highly variable passenger demand between months and even days. Passenger demand can range from 30 to 175 bookings a day. Sefofane's flight schedule changes daily because passengers can request a flight between any two locations, and still impose their own timing constraints for the departure and arrival of the aircraft. This usually takes place a few days or even hours before the passengers flight. Therefore, the aircraft do not operate over fixed routes or on a fixed schedule.

The air transportation problem we study here has similarities with the problem studied by Espinoza et al. [3]. However, there are differences, making the Sefofane problem unique in nature. These differences lie in the number of airstrips they use and the use of international airstrips. For example, Espinoza et al. [3] uses private and public landing strips in the United States, whereas Sefofane is limited to 21 private landing strips in the Okavango Delta. In addition, Sefofane only has one international airport which it transports passengers to and from safari camps. In [3] each aircraft has a home base and needs return to it at the end of each day. However, Sefofane's aircraft are allowed to stay the night at their last visited destination. Lastly, by governmental regulations customers are not allowed to use a car to travel in the Okavango Delta. This creates a unique demand scenario for the air transportation within the Okavango Delta.

Unlike other on-demand air transportation scheduling problems, the problem considered has many unique features. These are due to geographical locations and

the governments legislation. Therefore, new solution approaches are needed to tackle the problem. Hence we have decided to study the problem.

From the identified key features mentioned above, we can conclude that the Sefofane's scheduling problem creates challenges such as, short notice new demand. Therefore, creating a flight schedule for Sefofane is not that simple. Designing a model to deal with the dynamics of the on-demand air transportation with minimal operational costs will form the core of our research.

The problem we study uses a set of daily bookings. We use a number of heuristics to create 'direct' and 'indirect' flight legs that cater for the entire booking list. We then create the time-space network for the mathematical model. The model is then solved ensuring the optimal selection of flight legs and optimal fleet assignment, making it a semi-integrated approach.

1.5 Organization of the dissertation

Chapter 2 presents a full description of the problem at hand. It covers the scheduling process of Sefofane and all its operational factors and constraints.

Chapter 3 provides a detailed illustration of a multi-commodity flow network structure and its features. By manipulating certain features of this flow network we create an aircraft flow network. This is followed by a discussion on how the aircraft network is embedded into the basic fleet assignment model. Furthermore, we discuss the improved fleet assignment model which uses 'time windows'.

Chapter 4 introduces the formulation of the multi-commodity flow model used to solve Sefofane's semi-integral model. This is the main contribution of this research, which is the fleet and flight leg assignment model using 'time windows'. This chapter includes a description of the heuristics developed, namely '***Direct***', '***Indirect***' and '***Extension***', to create 'direct' and 'indirect' flight legs. We then use these heuristics to create an aircraft flow network, with the aid of an example. Finally, we describe

how the mathematical model is formulated using the time-space network.

In Chapter 5 we discuss the numerical results. We present the flight schedules for different sizes of booking lists. In addition, we present various aspects of the solutions obtained by our model.

Chapter 6: Concluding remarks are made including the merits and demerits of our model.

Chapter 2

Problem description

In this chapter, we present Sefofane's scheduling problem. We describe the problem and present simple scenarios to illustrate the need for optimization techniques. We begin with the background to the problem.

Sefofane's scheduling problem arose in a charter airline, Sefofane, operating in remote parts of Southern Africa. Sefofane was founded in 1991 by Neil and Suzy Lumsden [26]. It operates from five different main hubs located in Botswana, Nambia, Zambia, and South Africa. However, these hubs work independently. The problem we will present in this chapter will only focus on Sefofane's scheduling problem difficulties experienced by the Sefofane situated in Botswana. Sefofane Botswana (thereafter Sefofane) is located in Maun, the fifth largest town in Botswana. It is where Sefofane's main hub is located. Maun is the headquarters of many other air charter operations which run trips into the Okavango Delta. The Okavango Delta is the world's largest inland delta, and is located in Botswana. The Okavango delta is the home of a variety of wildlife which has become a very popular tourist attraction. Situated in this delta are a number of safari camps (a camp or lodge located within a large area of land where wild animals roam freely), which cater for small groups of tourists interested in viewing the wildlife. A map of the network of safari camps in the Okavango is provided in Figure 2.1.

Figure 2.1: A map of the Okavango Delta

Figure 2.1 presents Sefofane's main hub and the safari camps in the Okavango Delta. These safari camps make up Sefofane's flight network. Sefofane's flight network consists of 21 locations (see Appendix I), where MUB is the only international connection i.e. MUB is an international airport. This results in 420 possible flight legs. The main hub MUB is abbreviated as MUB (see Appendix I for abbreviations of the various camps). At each of these safari camps there are camp operators that cater for each tourist's requested itinerary. A tourist's itinerary is made up of guided tours to view the wildlife. In this flight network a flight leg is seen as flying from a specific departure camp to a specific arrival camp, at a given departure and arrival time. For example flying from Maun to Xigera in Figure 2.1 is considered as a flight leg.

The main task of Sefofane is to transport tourists with a light aircraft to and from these various exclusive safari camps in the Okavango Delta. A tourist or a group of tourists can fly directly anywhere in the network, and at anytime they wish. This motivates why Sefofane is an on-demand air transportation service. An on-demand air transportation service allows a customer (tourists) the opportunity to book a flight days or even hours before their flight. Above all the customer can choose a departure and arrival location, as well as a departure and arrival time. Therefore, an on-demand air transportation service creates challenges such as short notice new demands and unscheduled maintenances.

A normal operating day at Sefofane begins with transporting tourists from the main hub, MUB, to the various safari camps shown in Figure 2.1. Tourists generally arrive in MUB via a flight (local or international) or with their own transportation arrangements. The day also includes tourists being flown by a Sefofane aircraft from their existing camp they stayed over the previous night, to another safari camp. Sefofane has to create daily flight schedules that satisfy all the tourists' booking criteria, whilst selecting the most profitable pilot and aircraft type for each flight leg to and from the safari camps. The aircraft and fleet assignment, and pilot

selection process is based on the aircraft's equipment capabilities and maintenance planning, and the pilot's qualifications. At present the entire scheduling process is done manually.

Given that the flight scheduling is done manually the outcome may not always be optimal, due to the fact that the daily passenger loads vary considerably. Another factor contributing to this, is that the booking criteria of the tourist must be satisfied (e.g. departure and arrival times etc.). In addition, the flight scheduling process is also dependent on factors such as the Sefofane scheduler's competence and skill, and the countless operational factors that need to be taken into account. This process is often iterative and puts stresses on the personal responsible for the daily flight scheduling at Sefofane.

The daily manual scheduling at Sefofane is currently done by capturing the tourists' bookings information, selecting the appropriate aircraft for the flight route (making sure all the important factors and booking criteria are taken into consideration), and allocating a pilot (taking all his/her duty limitations into account). Once an aircraft and pilot have been chosen, the scheduler then looks at the booking list to figure out which route best seems to move the tourists to their desired destinations. There are three dedicated personnel employed by Sefofane for the scheduling purpose. Their jobs on a particular day are to prepare booking lists for the next day and facilitate in the execution of the flight schedule for the day.

2.1 Components of Sefofane's flight scheduling and operating factors

The operating factors presented below need to be considered when setting up flight schedules for tourists, aircraft and pilots.

2.1.1 The booking list

A tourist or group of tourists book a flight or a flight leg[1], by calling Sefofane with their desired booking request only days or even hours ahead of their flight. A number of tourists traveling together are bundled under one group name, typically family names or tour codes given by Sefofane. Every booking is grouped with the following underlying information: a departure location (***From***); an arrival location (***To***); the number of passengers in the group (***Pax***); a ***Group Name***; the earliest and latest departure time at the origin (***Etd*** and ***Ltd*** respectively); and the earliest and latest arrival time at the destination (***Eta*** and ***Lta*** respectively). This constitutes a booking list, and is done by the person at Sefofane responsible for organising and capturing the bookings. Table 2.1 contains an example of a booking list (not the complete daily booking list). It also presents in it's last column the booking index (***BI***). For example, the second row of Table 2.1 presents the booking information of group Boniello. It is a one member group (***Pax*** is 1), departing MUB (with an ***Etd*** of 6:00 and a ***Ltd*** of 18:00) and arriving at BBK (with an ***Eta*** of 6:00 and a ***Lta*** of 18:00), and is represented by the booking index 1.

We also present Table 2.2 a summary of the booking list in Table 2.1. Column 1 of Table 2.2 presents all five distinct safari camps presented in Table 2.1 and columns 2 and 3, respectively, present the groups departing from and arriving at the corresponding camps in column 1. For example, in row 2 five groups are departing MUB while no groups are arriving at MUB. Observe that there are no groups arriving in MUB, as we have chosen flight bookings that do not have MUB as an arrival location, i.e. no flights are arriving at MUB. Similarly, no flights are departing from XOR.

[1]A flight from a specific departure location to an arrival location, at a given departure and arrival time.

Table 2.1: An example of a daily reservation list

From	*To*	*Pax*	*Group Name*	*Etd*	*Ltd*	*Eta*	*Lta*	*BI*
MUB	BBK	1	Boniello	6:00	18:00	6:00	18:00	1
MUB	XOR	2	Dillion	6:00	18:00	6:00	18:00	2
BBK	XOR	1	Fisher	13:00	16:00	6:00	18:00	3
MUB	XIG	2	Norris	6:00	18:00	6:00	18:00	4
XIG	BBK	2	Rohde	11:00	16:00	6:00	18:00	5
MUB	BBK	1	Bledsoe	8:00	18:00	6:00	18:00	6
MUB	XOR	2	Bjorkman	6:00	18:00	6:00	18:00	7
BBK	SLI	2	Kate	6:00	18:00	6:00	18:00	8
BBK	XOR	1	Marshall	8:00	18:00	6:00	18:00	9
SLI	XOR	2	Khama	6:00	18:00	6:00	18:00	10
SLI	XOR	1	Speiler	6:00	18:00	6:00	18:00	11
BBK	XOR	1	Mark	8:00	18:00	6:00	18:00	12
XIG	BBK	1	Lewis	11:30	16:00	6:00	18:00	13

Table 2.2: Summary of the booking list Table 2.1

Safari camp	*Groups departing*	*Groups arriving*
MUB	Boniello, Dillion, Norris Bledsoe, Bjorkman	
BBK	Fisher, Kate, Marshall Mark	Boniello, Rhode, Bledsoe, Lewis
XIG	Rohde, Lewis	Norris
SLI	Khama, Speiler	Kate
XOR		Dillion, Fisher, Bjorkman, Marshall Khama, Speiler, Mark

2.1.2 Passenger type

Sefofane considers three types of passengers. These are the fare paying passengers, staff who travel for free and freight (see section 2.1.5 for the description of freight). Staff can be moved anytime but must be moved immediately after their request has been made. The key issue with staff is that they should not create extra aircraft flying hours[2] unless this is requested. Extra flying hours are created when an aircraft is solely dedicated to a passenger. Hence, an aircraft transporting a staff member

[2]The number of hours the aircraft spends flying from one camp to another.

needs to contain fare paying passengers as well.

2.1.3 The earliest and latest departure and arrival times

Every group of tourists in the booking list has an earliest and latest departure (arrival) time. These are used to determine the earliest and latest time a passenger can leave (arrive) a point of departure (arrival). Associated with these are two time intervals which are important to Sefofane's operational purpose. The first one is known as the turnaround time and the second one as the inter-camp transfer interval.

The turnaround time is the time between the time the aircraft arrives at a location and the time the aircraft departs from a location. In our case, when an aircraft lands at a camp it is given 10 minutes between the time of arrival and departure, which gives the passengers time to exit the aircraft and collect their luggage. It also gives the pilot time to load the new passengers and position the aircraft for take-off. This is called the turnaround time but is also known as the take-off and landing adjustment time.

The safari camps have certain time restrictions imposed on when inter-camp transfers can be done. As tourists want morning tea and an afternoon game drive at their arriving camp. Thus, the inter-camp flight time interval is typically from 06:00 through to 18:00. This means a tourist cannot depart (arrive) at a camp earlier than 06:00 or later than 18:00.

2.1.4 The maximum number of intermediate stops

Sefofane has a policy which states that a tourist can have a maximum number of three intermediate stops between the departure and arrival location of a tourist's booking.

2.1.5 Freight

The freight Sefofane receives, needs to be transported by a Sefofane aircraft to the requested safari camp. This freight is recorded as a booking under a group name, which is then booked onto an aircraft. It is therefore easier to treat freight as a fare paying passenger. Hence, in our implementation group names are assigned to the freight.

2.1.6 Aircraft type, maintenance and refueling

There are only three aircraft types (three fleet) available namely the Cessna 206 (4 passenger seats), the Pilatus PC-12 (8 passenger seats) and the Cessna 208 (10 passenger seats) also known as a caravan, and are abbreviated as C206, PC12 and C208, respectively. Each aircraft has a seat and weight capacity, which limits the number of passengers that can be accommodated. Hence, each aircraft type is different in it's characteristics and better suited to different types of trips and different group sizes.

Maintaining the aircraft to make sure that all it's instruments and equipment are in good condition is one of the most important features in managing an airline company. Therefore, each aircraft needs to be checked into the hangar[3] for a regular service. These services are made up of oil refills, engine and propeller repairs etc., otherwise known as maintenance. At Sefofane, each aircraft is checked into the hangar after it has accumulated a total of 50 and 100 flying hours (hr). The 50 flying hour check takes about 2 hours and the 100 flying hour takes up to 3 days. When the scheduler is preparing a schedule he/she needs to make sure that the aircraft chosen does not have to go in for maintenance or will not exceed it's maintenance interval on any flight leg of the flight schedule of the day. So, the significance of maintenance routing is to ensure that all the aircraft do not exceed their maintenance interval

[3]A garage in which the aircrafts are serviced in.

before taking off on the last flight leg back to the main hub, MUB, or their final arrival location.

Finally, we describe how refueling affects the scheduling of an aircraft. An aircraft needs sufficient fuel reserves to arrive at it's destination. So if it is necessary to refuel, then at least one of its stops needs to be a refueling station. The effects of refueling on the schedule is that fuel is only available at certain camps and not all the aircraft need to refuel after the same amount of flying hours. For instance a C206 can fly for 3.5 hours before refueling, whereas a C208 can fly for 6 hours before refueling. In addition, all the aircraft do not use the same fuel, some use Avgas and others use Jet A1. If the last flight leg for a day requires the aircraft to refuel, then the camp in which the pilot and aircraft pair are going to stay overnight needs to be a refueling station.

2.1.7 The pilots

There are different levels of pilot, some pilots are qualified to fly the C206, PC12 and C208 whereas others are in training and can only fly two or one of these aircraft types. Every flight schedule needs to be designed to satisfy the pilot's duty time and daily flight hour limitations[4]. A pilot's maximum daily flight hour limitation is 10 hours (which they can not go beyond). They also fly for 6 days in a row and get the 7th day off. Given that the pilot and aircraft pair can stay overnight for a maximum of two nights only, the scheduler again needs to make sure the pilot returns to MUB before his/her day off. Therefore, they must return to the main hub MUB on the night before their day off. The pilot and aircraft endure the entire flight together, and are regarded as a pair until the aircraft has landed in the main hub. Once they have landed in the main hub, they are separated and no longer regarded as a pair.

[4]The number of hours a pilot can fly per day.

2.2 The manual schedule at Sefofane

When planning a schedule, the scheduler needs to yield a set of decisions. The scheduler has three lists to make his/her decisions from. These are the list of bookings (see Table 2.1), pilots and aircraft. In Table 2.3 we present examples of the aircraft used by Sefofane and their specifications. Table 2.4 presents the pilots and their specifications. The scheduling is done manually on a daily basis and the sequence of decisions which make up this process are made as follows:

Step 1: Choosing the best possible aircraft.

Step 2: Picking an optimal pilot.

Step 3: Selecting a set of bookings (in the form of groups) from the set of bookings of the day.

Step 4: Figuring out a suitable route for the set of bookings chosen in **Step 3.**

For each group in the list of bookings, the Sefofane scheduler needs the following information to create each flight leg. These are the earliest departure time (***Etd***), latest departure time (***Ltd***), earliest arrival time (***Eta***), latest arrival time (***Lta***), departure location (***From***), arrival location (***To***), the number of passengers flying on a specific flight leg (***Pax***) and the names of these groups. ***Etd***, ***Ltd***, ***Eta*** and ***Lta*** are used to ensure that passengers are not scheduled to leave (arrive) too early or too late at their departure (arrival) camp. ***Pax*** is used to make sure that the number of passengers does not exceed the aircraft's seating capacity. Lastly, ***From*** and ***To*** indicates where the passenger(s) needs to be picked up and dropped off. It is vital that all of these are recorded correctly.

The decisions made in **Steps 1** to **4** are all crucial, as the resulting operating costs can be extremely high, due to an incorrect decision. Next, we present a small sample problem to illustrate how these steps are used to create a flight schedule.

2.3 A sample problem

To set up a sample problem the following lists are needed. A small booking list, a list of available pilots and their qualifications, and a list of available aircraft and their specifications. For the sample problem we will make use of Tables 2.1, 2.3 and 2.4. The daily flight time limitations for a pilot as well as the maintenance requirements are ignored in the presentation of the sample problem. We assume that the sample problem presented always has a feasible solution.

We have presented features of a daily booking list using Table 2.1. In Table 2.3 we present the specifications of the aircraft used. These are as follows: aircraft's name (***AN***); aircraft's fleet type (***AFT***); aircraft's starting location (***ASL***); aircraft's speed (***AS***) in km/hr; aircraft's seating capacity (***ASC***); aircraft's cost[5] (***AC***) in dollars (\$)/hr; aircraft's fuel cost (***AFC***) in \$/hr; aircraft's overnight fee[6] (***AOF***) in \$; take-off adjustment time (***TAT***) in minutes (min); landing adjustment time (***LAT***) in min and aircraft's landing fee (***ALF***) in \$.

In Table 2.4 we present a list of all the available pilots and their specifications which includes the following: pilot's name (***PN***); pilot's starting location (***PSL***); aircraft types each pilot is qualified to fly (***APQ***); pilot's flying cost (***PFC***) in \$/hr and pilot's overnight fee (***POF***) in \$.

The scheduler now follows the 4 steps listed earlier, and constructs a flight schedule for the day using Tables 2.1, 2.3 and 2.4. Once each of the passenger groups in Table 2.1 are assigned to a flight leg which has a pilot and an aircraft, the scheduler's job is done and the flight schedule is complete.

We now create a manual flight schedule of the scheduling problem presented in Tables 2.1 - 2.4 using the four steps discussed earlier. The details of this manual schedule are presented in Table 2.5 and Figure 2.2. All six ***Flight legs*** created for this schedule are shown in Figure 2.2, which are denoted by an arrow going from the

[5]The flying cost of the aircraft per hour.

[6]The fee paid for an aircraft, pilot or both to stay at a camp for the night.

Table 2.3: Set of available aircraft and their specifications

AN	***AFT***	***ASL***	***AS*** *km/hr*	***ASC***	***AC*** *$/hr*	***AFC*** *$/hr*	***AOF*** *$*	***TAT*** *min*	***LAT*** *min*	***ALF*** *$*
A2-ADK	C206	MOM	260	5	500	210	250	5	5	5
A2-AIV	C206	CTB	260	5	500	210	250	5	5	5
A2-ANT	C206	HND	260	5	500	210	250	5	5	5
A2-BEE	C206	SLI	260	5	500	210	250	5	5	5
A2-BOK	C206	XIG	260	5	500	210	250	5	5	5
A2-BUF	C208	XOR	280	12	500	275	250	5	5	5
A2-EGL	C208	CBE	280	12	500	275	250	5	5	5
A2-GNU	C208	MUB	280	12	500	275	250	5	5	5
A2-FOX	PC12	MUB	440	8	900	300	500	12	12	0
A2-JKL	C206	BBK	260	5	500	210	250	5	5	5
A2-LEO	C208	MUB	280	12	500	275	250	5	5	5
A2-OWL	C206	VUM	260	5	500	210	250	5	5	5
A2-ZEB	C208	JAO	280	12	500	275	250	5	5	5

AN: Aircraft name; ***AFT***: Aircraft fleet type; ***ASL***: Aircraft starting location; ***AS***: Aircraft speed; ***ASC***: Aircraft seating capacity; ***AC***: Aircraft cost; ***AFC***: Aircraft fuel cost; ***AOF***: Aircraft overnight fee; ***TAT***: Take-off adjustment time; ***LAT***: Landing adjustment time; ***ALF***: Aircraft landing fee.

departing camp to the arrival camp. Each ***Flight leg*** is also labeled with a flight leg number corresponding to the ***Flight legs*** in Table 2.5, column 1. In addition, a list of booking indices ***BI*** (see Table 2.1) associated with each ***Flight leg*** are shown below the flight leg in Figure 2.2. It can be seen that all the groups in the booking list presented in Table 2.1 are catered for in this flight schedule as shown in Figure 2.2.

The additional features of this manual schedule are presented in Table 2.5 (such as the aircraft and pilot pair used, the overnight information (***OI***), and the departure and arrival time of each ***Flight leg***). There are three ***Flights*** (shown in three parts in Table 2.5, with a pilot and aircraft assigned to them) each consisting of two ***Flight legs*** as shown in Table 2.5. Each ***Flight*** consists of the following: an aircraft; a pilot; two ***Flight legs***; ***Etd***; ***From***; ***Eta***; ***To***; ***Pax***; ***Group*** (where the groups in italics denote that the group will get off at the end of the ***Flight leg***, and the groups in bold will remain in the aircraft and embark on the next ***Flight leg***). In Table

Table 2.4: Set of available pilots and their specifications

PN	***PSL***	***APQ***			***PC*** ***$/hr***	***POF*** ***$***
Nick	MOM	PC12	C206	C208	1200	120
Andrew	CTB	PC12	C206	C208	1200	120
Warren	HND	PC12	C206	C208	1200	120
Quinton	SLI	PC12	C206	C208	1200	120
Gavin	XIG	C206	PC12		1200	120
Nic	XOR	C206	PC12	C208	1200	120
Graham	CBE	C206	C208		1200	120
Theo	MUB	C206	C208	PC12	1200	120
Aleix	MUB	C206	C208	PC12	1200	120
Gustav	BBK	C206	C208	PC12	1200	120
Iain	MUB	C206	C208	PC12	1200	120
Alexi	VUM	C206	C208	PC12	1200	120
Pillip	JAO	C206	C208		1200	120

PN: Pilot name; ***PSL***: Pilot starting location; ***APQ***: Aircraft types pilot is qualified to fly; ***PC***: Pilot cost; ***POF***: Pilot overnight fee.

2.5 the final destination of each passenger is denoted by ***FD*** (the final destination of each passenger on a specific ***Flight leg*** is not always necessarily the same as the *Flight leg's* arrival locations) and the overnight information by ***OI***. We now discuss the details of ***Flight 1*** in Table 2.5. However, ***Flights 2*** and ***3*** can be understood in a similar way.

For ***Flight 1*** we have chosen aircraft A2-LEO (Table 2.3, row 12), which is a C208 and is positioned in MUB. There are three pilots which are situated in MUB, and who are qualified to fly the aircraft type C208 namely: Theo, Aleix and Iain (Table 2.4, rows 9, 10 and 12 respectively). From these three pilots we have selected Iain (Table 2.4, row 12). Passenger groups Boniello, Dillion, Bjorkman and Fisher have been picked up to be transported to their destinations by ***Flight 1*** (see Tables 2.1 and 2.2). ***Flight 1*** has two ***Flight legs***. ***Flight leg*** 1 departs MUB at 9:10 to arrive in BBK at 10:20 (see Table 2.5, row 3, columns 1 to 5). This ***Flight leg*** has Boniello, Dillion and Bjorkman on it (see Table 2.5, rows 3 to 5, column 7 respectively). Passenger group Boniello is then dropped off at BBK,

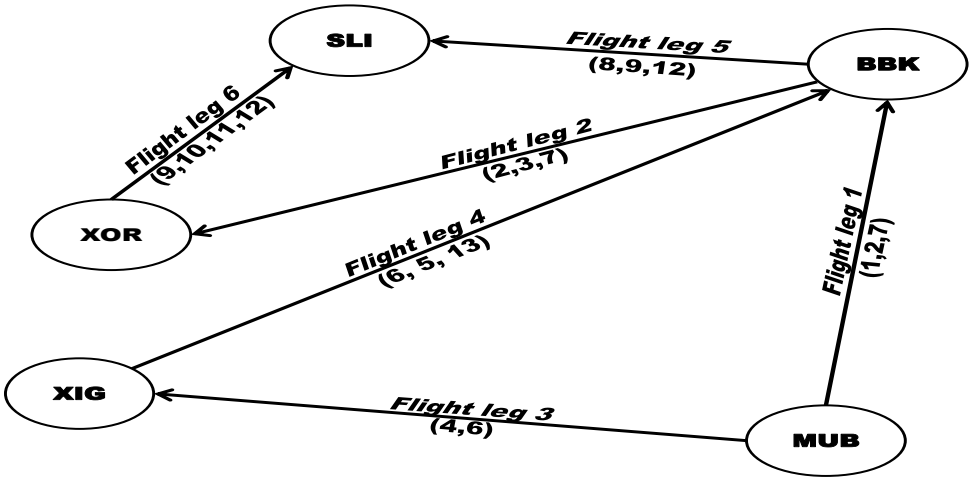

Figure 2.2: Psuedo-network of the solution in Table 2.5

and Fisher is loaded for ***Flight leg*** 2 (see Table 2.1, rows 2 and 8, respectively for bookings details and Table 2.5, row 8, column 7 for ***Flight leg*** 2 details). Dillion and Bjorkman remain on the aircraft while this change takes place. ***Flight leg*** 2 has passenger groups Dillion, Bjorkman and Fisher on it, which will be flown to XOR from BBK (see Table 2.5, rows 6 to 8 respectively). Notice that the passenger group Fisher has an ***Etd*** of 13:00 (see Table 2.1, row 4, column 5). Observe that this ***Flight leg*** departs BBK at 13:00 and lands in XOR at 13:55 (see Table 2.5, row 6, columns 3 to 5 respectively). This automatically satisfies Fisher's booking since they can only be loaded at 13:00 and later. Notice also that the total passenger load for ***Flight legs*** 1 and 2 is 5 (see Table 2.5, rows 3 to 8, column 6 respectively), which makes the aircraft type C208 suitable with regards to it's maximum seating capacity of 12 (see Table 2.3, row 11, column 7). The overnight information (***OI***) is given at the bottom of ***Flight 1***.

2.3.1 Flight time and cost calculation

The three ***Flights*** presented in Table 2.5 have their associated costs. We now present the cost calculation of these ***Flights***. We show the cost calculation for ***Flight 1*** only. The other ***Flights'*** cost can be calculated similarly. We present the cost of all three ***Flights*** in Table 2.6. We denote the total distance traveled by Dt, total flight duration by Ft. The cost calculation of ***Flight 1*** is as follows:

To calculate the exact distance of each of the ***Flight legs*** that make up ***Flight 1*** i.e. MUB $\rightarrow$ BBK and BBK $\rightarrow$ XOR, we use their GPS co-ordinates given in degrees minutes seconds (see Appendix I). We now demonstrate how this can be done by calculating the distance of the ***Flight leg*** MUB $\rightarrow$ BBK as an example. The distances of the other flight leg can be calculated similarly. The respective (longitude, latitude) GPS co-ordinates for location MUB and location BBK are (23°25' 52", -19°58' 21") and (25°9' 44",-17°49' 58"). Firstly, we need to convert each of these co-ordinates into degrees. We use the GPS co-ordinates of location MUB to show this conversion. We denote the co-ordinates in degrees as ($Long$, Lat). The conversion is as follows:

$$\begin{aligned}(Long, Lat) &= \left(|deg_o| + \frac{|min_o|}{60 + \frac{|sec_o|}{3600}}, |deg_a| + \frac{|min_a|}{60 + \frac{|sec_a|}{3600}}\right), \\ &= \left(|23| + \frac{|25|}{60 + \frac{|52|}{3600}}, |-19| + \frac{|58|}{60 + \frac{|21|}{3600}}\right), \\ &= (23.4^{\text{o}}, -19.9^{\text{o}}),\end{aligned}$$

where deg_o (deg_a) denotes the number of degrees in the longitude (latitude), min_o (min_a) denotes the number of minutes in the longitude (latitude), and sec_o (sec_a) denotes the number of seconds in the longitude (latitude). The ($Long$, Lat) co-ordinates for location XIG can be calculated similarly and are given as (25.2°, -17.8°).

We use the ($Long$, Lat) of both locations to calculate the distance between them. We denote the distance between location a and location b as $dist(a, b)$. The distance

can be written as follows:

$$\begin{aligned} dist(a,b) &= deg2km(\sqrt{(Long_b - Long_a)^2 + (Lat_b - Lat_a)^2}), \\ dist(MUB,BBK) &= deg2km(\sqrt{(25.2^o - 23.4^o)^2 + (-17.8^o - (-19.9^o))^2}), \\ &= deg2km(\sqrt{0.64^o + 0.64^o}), \\ &= deg2km(2.97)^o, \\ &= 306\ km, \end{aligned}$$

where $Long_a$ ($Long_b$) is the longitude degrees of location a (b), Lat_a (Lat_b) is the latitude degrees of location a (b) and $deg2km$ is a Matlab function which converts degrees into km. The distances of all the other possible flight legs in Sefofane's flight network are given in the distance matrix in Appendix II. We use the distances of each of the flight legs to calculate Dt of ***Flight 1***.

$$\begin{aligned} Dt &= dist(\text{MUB}, \text{BBK}) + dist(\text{BBK}, \text{XOR}) \\ &= 306\ km + 250\ km, \\ &= 556\ km, \end{aligned}$$

where $dist(a,b)$ denotes the distance from location a to location b. The total flying time is given by:

$$\begin{aligned} Ft &= \frac{Dt}{AS} \\ &= \frac{556\ km}{280\ km/hr}, \\ &= 1.9857\ hr, \end{aligned}$$

where AS denotes the aircraft speed (equal speed for both ***Flight legs***) and the value of which has been taken from Table 2.3.

Finally, we calculate the aircraft and pilot cost of ***Flight 1***. We denote the total aircraft cost by Tac and total pilot cost by Tpc. In ***Flight 1*** the aircraft used was

A2-LEO (C208). In this ***Flight*** the pilot and aircraft pair stayed overnight (see Table 2.5 row 9) incurring overnight cost (***AOF***). The aircraft landed in two camps resulting in two aircraft landing fees (***ALF***). The total aircraft cost can be written as:

$$\begin{aligned} Tac &= Ft(AC + AFC) + 2(ALF) + AOF \\ &= 1.9857\ hr(\$500\ /hr + \$275\ /hr) + 2(\$5) + \$250, \\ &= \$1798.92, \end{aligned}$$

where the values of ***AC*** (aircraft cost), ***AFC*** (aircraft flying cost), ***ALF*** and ***AOF*** can be found in row 12 Table 2.3. The pilot Iain was used in this ***Flight*** and from Table 2.4 it can be seen that pilot costs (***PC***) is \$1200 and the pilot overnight fee (***POF***) \$120. Therefore the total pilot cost is given by:

$$\begin{aligned} Tpc &= Ft(PC) + POF \\ &= 1.9857\ hr(\$1200\ /hr) + \$120 \\ &= \$2502.84. \end{aligned}$$

The overall cost is calculated below which is presented in Table 2.6.

$$\begin{aligned} Costs &= Tac + Tpc \\ &= \$4301.76. \end{aligned}$$

2.4 Effects of optimization in flight scheduling

Having shown how a flight schedule can be created using a list of bookings (e.g. Table 2.1), a list of aircraft (e.g. Table 2.3) and a list of pilots (e.g. Table 2.4), we now show how the schedule presented in Table 2.5 can be optimized. For this, we consider ***Flights 1***, ***2***, and ***3*** presented in Table 2.5. We create three scenarios by

(a) changing the aircraft and the route, (b) altering both the pilot and the route, and (c) changing the pilot and aircraft. These are presented below.

Scenario 1: Changing the route and the aircraft

We present Table 2.7 and Figure 2.3 a new feasible solution created by changing the route and the aircraft of our original solution (presented in Figure 2.2 and Table 2.5). It can be seen in Figure 2.3 that this solution only has four ***Flight legs***. This solution caters for all the bookings in Table 2.1. The additional features are presented in Table 2.7 (such as the aircraft and pilot pair used, the overnight information, and the departure and arrival time of each ***Flight leg***).

The aircraft A2-GNU (Table 2.3, row 9) was chosen to fly this route (Table 2.7 row 1), as the number of passengers on each ***Flight leg*** are above 7 (see Table 2.7, column 6). The corresponding pilots for aircraft A2-GNU are Iain, Theo and Aleix. As they are all qualified to fly aircraft type C208, and are positioned in MUB i.e. the starting location of aircraft A2-GNU. Iain was decided on for this ***Flight*** (see Table 2.7, row 2). The departure and arrival time of each ***Flight leg*** is given in Table 2.7, columns 2 and 4, respectively. The ***Etd***, ***Ltd***, ***Eta*** and ***Lta*** of each booking in Table 2.1 are satisfied by these times. Hence, this is a feasible solution with respect to the timing constraints of each booking.

The total cost for this flight was calculated and is given as \$5322.94, where $dist(MUB, XIG) = 101\ km$, $dist(XIG, BBK) = 320\ km$, $dist(BBK, SLI) = 201\ km$, $dist(SLI, XOR) = 49\ km$, $AS = 280\ km/hr$, $Ft = 2.3964\ hr$, $AC =$ \$500 per hr, $AFC =$ \$275 per hr, $ALF =$ \$5, $AOF =$ \$250, $PC =$ \$1200 per hr, $POF =$ \$120, $Tac =$ \$2127.23 and $Tpc =$ \$3195.71 . Clearly, the total cost of this schedule is much lower than the total cost (\$10312.99) incurred in Table 2.6.

Scenario 2: Changing the pilot and the route

Next, we present Table 2.8 and Figure 2.4 a solution created by changing the pilot and the route of ***Flights 1*** and ***2*** in the old solution given in Table 2.5, and keeping ***Flight 3*** the same. Additional features are presented in Table 2.8 (such as the pilot

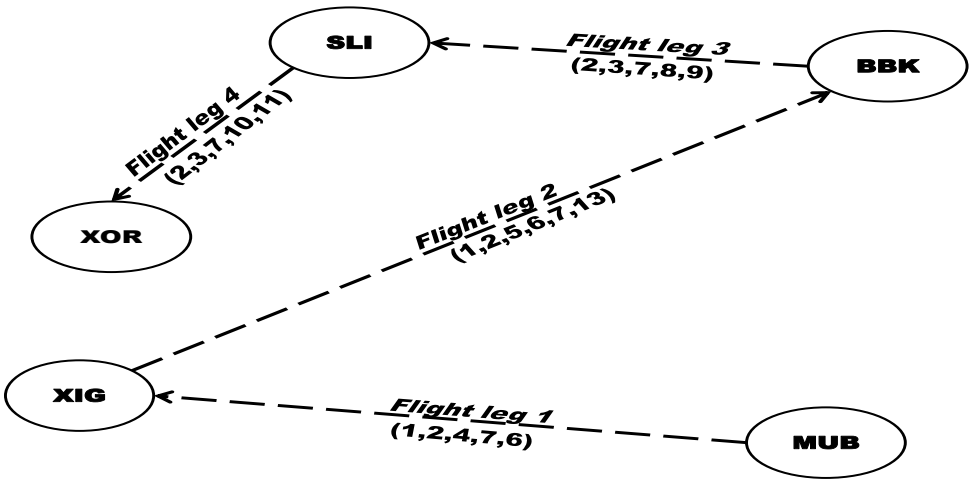

Figure 2.3: Psuedo-network of the solution of Scenario 1

and aircraft pair, the departure and arrival time of each *Flight leg*, and the overnight information for the ***Flight***). The ***Etd***, ***Ltd***, ***Eta*** and ***Lta*** of each booking in Table 2.1 flown on this ***Flight*** are satisfied. Hence, this is a feasible solution with respect to the timing constraints of the bookings.

The cost of this ***Flight*** including the cost of ***Flight 3*** (which remains the same) is calculated as \$7732.81, where $dist(MUB, XIG) = 101\ km$, $dist(XIG, BBK) = 320\ km$, $dist(BBK, XOR) = 250\ km$, $AS = 280\ km/hr$, $Ft = 2.3964\ hr$, $AC =$ \$500 per hr, $AFC =$ \$275 per hr, $ALF =$ \$5, $AOF =$ \$250, $PC =$ \$1200 per hr, $POF =$ \$120, $Tac =$ \$2122.23 and $Tpc =$ \$3075.71. This cost is also lower than the total cost (\$10312.99) reported in Table 2.6.

Scenario 3: changing the pilot and the aircraft

Lastly, we present Table 2.9 and Figure 2.5 a new solution to ***Flight 3*** in Table 2.5. This new solution was created by changing the pilot and the aircraft of ***Flight 3***. The other ***Flights*** in Table 2.5 remain the same.

The aircraft selected was aircraft A2-ADK (see Table 2.9 row 1). This aircraft is type C206, whereas the previous aircraft A2-LEO is type C208. Take note that

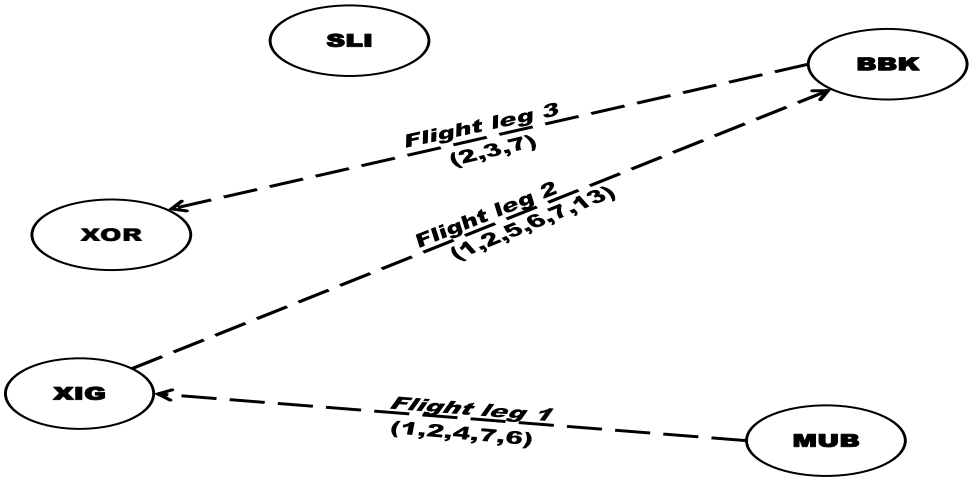

Figure 2.4: Psuedo-network of the solution of Scenario 2

aircraft A2-ADK is not positioned in MUB but MOM. Furthermore, the departure location of ***Flight leg*** 1 of ***Flight 1*** is MUB, and for this reason the new aircraft (A2-ADK) will have to fly from MOM to MUB in order to perform ***Flight leg*** 1 of ***Flight 1***, this will now be *Flight leg* 1 of the new ***Flight*** (see Table 2.9, row 4). This is a new ***Flight leg*** along with the existing ***Flight legs*** of ***Flight 1*** in Table 2.5. The only pilot suitable for this aircraft is Nick, as he is situated in MOM (see Table 2.4 row 2 column 2). Therefore we have chosen him to fly the aircraft (shown in Table 2.9 row 2). The ***Etd***, ***Ltd***, ***Eta*** and ***Lta*** of each booking in Table 2.1 flown are satisfied on this ***Flight*** are not violated, this is a feasible solution with respect to the timing constraints.

The cost of this ***Flight*** plus the costs of ***Flights 2*** and ***3*** (given in Table 2.6 rows 2 and 3 column 4) is \$11153.56, where $dist(MOM, MUB) = 110\ km$, $dist(MUB, BBK) = 306\ km$, $dist(BBK, XOR) = 250\ km$, $AS = 260\ km/hr$, $Ft = 2.4087\ hr$, $AC =$ \$500 per hr, $AFC =$ \$210 per hr, $ALF =$ \$5, $AOF =$ \$250, $PC =$ \$1200 per hr, $POF =$ \$120, $Tac =$ \$2131.81 and $Tpc =$ \$3210.54. This cost is higher than the total cost (\$10312.99) presented in Table 2.6.

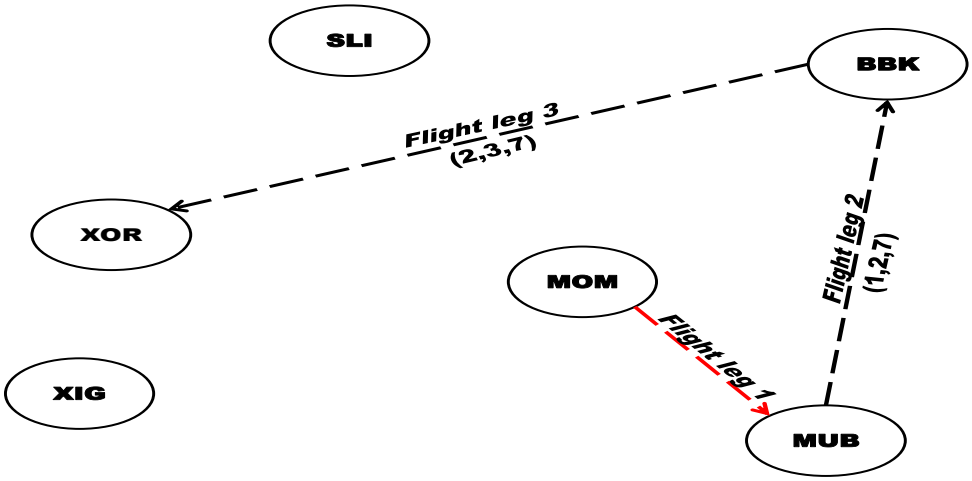

Figure 2.5: Psuedo-network of the solution of Scenario 3

2.5 The Sefofane scheduling problem

The sample problem presented earlier considers many inherent optimization sub-problems (as we have demonstrated in section 2.3). For example, optimal aircraft and pilot selection as well as flight leg selection. However, for the purpose of this research we address the two main components of the problem. In particular, we solve the optimal flight leg selection and fleet assignment jointly. Unlike commercial airlines, our problem caters for the entire satisfaction of the demands in the daily booking list. This problem will, hereafter, be referred to as Sefofane's scheduling problem.

Table 2.5: A schedule for the sample problem

Flight 1; ***Aircraft***: A2-LEO; ***Pilot***: Iain							
Flight leg	***Etd***	***From***	***Eta***	***To***	***Pax***	***Group***	***FD***
1	9:10	MUB	10:20	BBK	1	*Boniello*	BBK
					2	**Dillion**	XOR
					2	**Bjorkman**	XOR
2	13:00	BBK	13:55	XOR	2	*Dillion*	XOR
					2	*Bjorkman*	XOR
					1	*Fisher*	XOR
OI for Flight 1: XOR is the last destination and therefore the aircraft and pilot will incur an overnight fee.							
Flight 2; ***Aircraft***: A2-FOX; ***Pilot***: Aleix							
Flight leg	***Etd***	***From***	***Eta***	***To***	***Pax***	***Group***	***FD***
3	11:15	MUB	11:30	XIG	2	*Norris*	XIG
					1	**Bledsoe**	BBK
4	11:40	XIG	12:25	BBK	1	*Bledsoe*	BBK
					2	*Rohde*	BBK
					1	*Lewis*	BBK
OI for Flight 2: BBK is the last destination and therefore the aircraft and pilot will incur an overnight fee.							
Flight 3; ***Aircraft***: A2-JKL; ***Pilot***: Gustav							
Flight leg	***Etd***	***From***	***Eta***	***To***	***Pax***	***Group***	***FD***
5	10:30	BBK	11:20	SLI	2	*Kate*	SLI
					1	**Marshall**	XOR
					1	**Mark**	XOR
6	11:30	SLI	11:45	XOR	1	*Marshall*	XOR
					1	*Mark*	XOR
					2	*Khama*	XOR
					1	*Speiler*	XOR
OI for Flight 3: XOR is the last destination and therefore the aircraft and pilot will incur an overnight fee.							

Etd: Earliest departure time; ***From***: Departure location; ***Eta***: Earliest arrival time; ***To***: Arrival location; ***Pax***: Number of passengers; ***Group***: Group name; ***FD***: Final destination; ***OI***: Overnight information.

Table 2.6: Cost of each flight in Table 2.5

Flight	***Dt*** (km)	***Ft*** (hr)	***Costs*** (\$)
1	556	1.9857	4501.78
2	421	0.95681	3396.34
3	250	0.96153	2414.87

Dt: Total distance; ***Ft***: Flight time.

Table 2.7: Flight schedule for Scenario 1

Aircraft: A2-GNU							
Pilot: Iain							
Flight leg	***Etd***	***From***	***Eta***	***To***	***Pax***	***Group***	***FD***
1	11:40	MUB	12:05	XIG	1	**Boniello**	BBK
					2	**Dillion**	XOR
					2	**Bjorkman**	XOR
					2	*Norris*	XIG
					1	**Bledsoe**	BBK
2	12:15	XIG	13:25	BBK	1	*Boniello*	BBK
					2	**Dillion**	XOR
					2	**Bjorkman**	XOR
					2	*Rohde*	BBK
					1	*Lewis*	BBK
					1	*Bledsoe*	BBK
3	13:35	BBK	14:20	SLI	2	**Dillion**	XOR
					2	**Bjorkman**	XOR
					1	**Fisher**	XOR
					2	*Kate*	SLI
					1	**Marshall**	XOR
4	14:30	SLI	14:45	XOR	2	*Dillion*	XOR
					2	*Bjorkman*	XOR
					1	*Fisher*	XOR
					2	*Khama*	XOR
					1	*Speiler*	XOR
OI for Flight 3: XOR is the last destination and therefore the aircraft and pilot will incur an overnight fee.							

Etd: Earliest departure time; ***From***: Departure location; ***Eta***: Earliest arrival time; ***To***: Arrival location; ***Pax***: Number of passengers; ***Group***: Group name; ***FD***: Final destination; ***OI***: Overnight information.

Table 2.8: Flights schedule for Scenario 2

Aircraft: A2-LEO							
Pilot: Theo							
Flight leg	***Etd***	***From***	***Eta***	***To***	***Pax***	***Group***	***FD***
1	11:30	MUB	11:55	XIG	1	**Boniello**	BBK
					2	**Dillion**	XOR
					2	**Bjorkman**	XOR
					2	*Norris*	XIG
					1	**Bledsoe**	BBK
2	12:05	XIG	13:15	BBK	1	*Boniello*	BBK
					2	**Dillion**	XOR
					2	**Bjorkman**	XOR
					2	*Rohde*	BBK
					1	*Lewis*	BBK
					1	*Bledsoe*	BBK
3	13:25	BBK	14:20	XOR	2	*Dillion*	XOR
					2	*Bjorkman*	XOR
					1	*Fisher*	XOR
OI for Flight 3: XOR is the last destination and therefore the aircraft and pilot will incur an overnight fee.							

Etd: Earliest departure time; ***From***: Departure location; ***Eta***: Earliest arrival time; ***To***: Arrival location; ***Pax***: Number of passengers; ***Group***: Group name; ***FD***: Final destination; ***OI***: Overnight information.

Table 2.9: Flight schedule for Scenario 3

Aircraft: A2-ADK							
Pilot: Nick							
Flight leg	***Etd***	***From***	***Eta***	***To***	***Pax***	***Group***	***FD***
1	12:35	MOM	12:25	MUB	0	Empty	MUB
2	12:35	MUB	13:45	BBK	1	*Boniello*	BBK
					2	**Dillion**	XOR
					2	**Bjorkman**	XOR
3	13:55	BBK	14:50	XOR	2	*Dillion*	XOR
					2	*Bjorkman*	XOR
					1	*Fisher*	XOR
OI for Flight 3: XOR is the last destination and therefore the aircraft and pilot will incur an overnight fee.							

Etd: Earliest departure time; ***From***: Departure location; ***Eta***: Earliest arrival time; ***To***: Arrival location; ***Pax***: Number of passengers; ***Group***: Group name; ***FD***: Final destination; ***OI***: Overnight information.

Chapter 3

The multi-commodity network flow formulation

In this chapter, we first present a flow network. We then present the multi-commodity network flow model for the fleet assignment problem. The mathematical model formulated to solve the flight scheduling problem, is an application of the multi-commodity network flow formulation. A multi-commodity network flow problem is a network flow problem with multiple commodities (goods) flowing through a flow network. Thus, a flow network forms the foundation of a multi-commodity network flow problem formulation. Next, we present the flow network.

3.1 The flow network

A flow network is a directed graph containing nodes and arcs which join the nodes together. We present a flow network structure and give a detailed description of it. We then present an aircraft fleet network structure.

3.1.1 The flow network structure

A flow network structure [33] allows a feasible flow through it. We present two examples of the flow network in Figures 3.1 and 3.2. Using these figures, we describe the features of the flow network. The first flow network is presented in Figure 3.1 and its features are presented below:

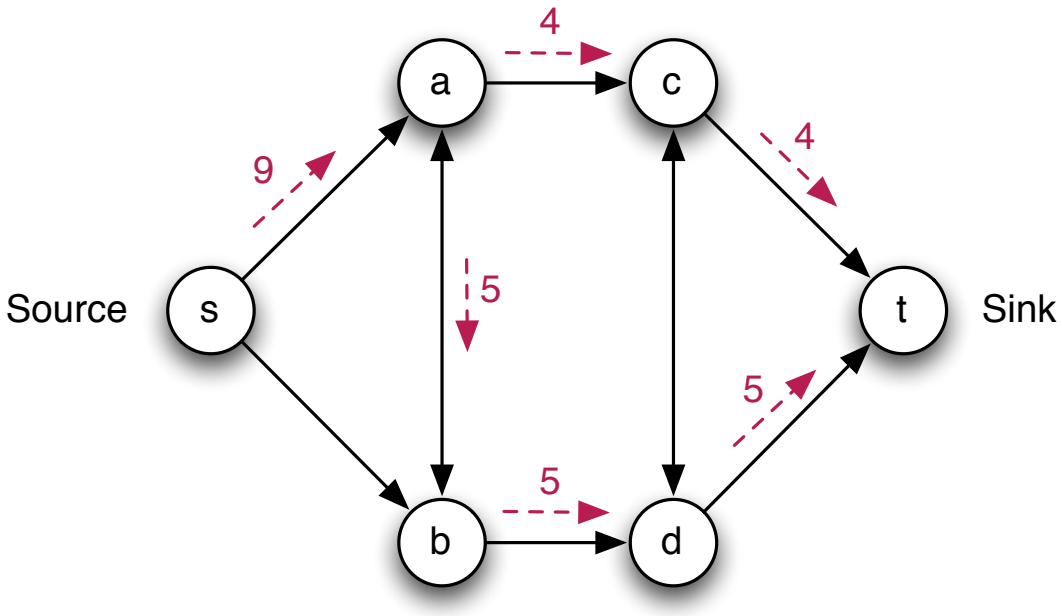

Figure 3.1: A flow network

Nodes - The nodes in a network represent key decisions taken at a specific 'place' and time, such as which commodity (good) needs to be 'transported' next. For example, there are four nodes labeled as **a**, **b**, **c** and **d** in Figure 3.1.

Arcs - The arcs represent the flow of 'transportation' from one node to the next. The arrows joining the nodes in Figure 3.1 represent the arcs. The small dashed arrow along each arc represents a particular feasible flow through the specific arc. For example, the arc from node **a** to node **c** has a flow of 4 flowing through it.

Conservation of flow - The flow throughout the network and at each node in the network must satisfy the conservation of flow constraint. This means that the amount of flow into a network (node) must be equal to the amount of flow out of the network (node). For example, in Figure 3.1 the amount of flow leaving node **a** (4 plus 5) equals the flow entering node **a** (9). Hence, the incoming flow and outgoing flow at node **a** are equal.

Source (sink) node - The source (sink) node marks the start (end) of the network. The source (sink) node is the exception to the conservation of flow constraint, as the source (sink) node only has outgoing (incoming) flow. For example, in Figure 3.1 the source (**s**) (sink (**t**)) node has a total outgoing (incoming) flow of 9.

Supersource (supersink) node - If more than one source (sink) node is required in the network, an additional supersource (supersink) node is added to compensate for this. The supersource (supersink) node now acts as the source (sink) node of the entire network. Figure 3.2 is an instance of a network with multiple source (sink) nodes and a supersource (**SS**) (supersink (**ST**)) node.

Wrap arc - The wrap arc conserves the flow exiting the source/supersource node and entering the sink/supersink node. It artificially joins these two nodes to ensure that the flow is conserved throughout the network. However, this type of arc is not used in all flow problems, only in cases where there is a cyclic flow. For example, in Figure 3.2 the wrap arc connects the supersource (**SS**) and supersink (**ST**) nodes together. Notice that the flow on this arc is the same as the total flow exiting the **SS** node and entering the **ST** node. This flow runs from the sink/supersink node to the source/supersource node.

Arc capacity - Each arc has a flow capacity. This limits the amount of flow that can travel along an arc.

Demand satisfaction - The flow through the network needs to satisfy a set of demands. In other words, all the required commodities (goods) must be 'transported' from their origin to their destination via the arcs in a network.

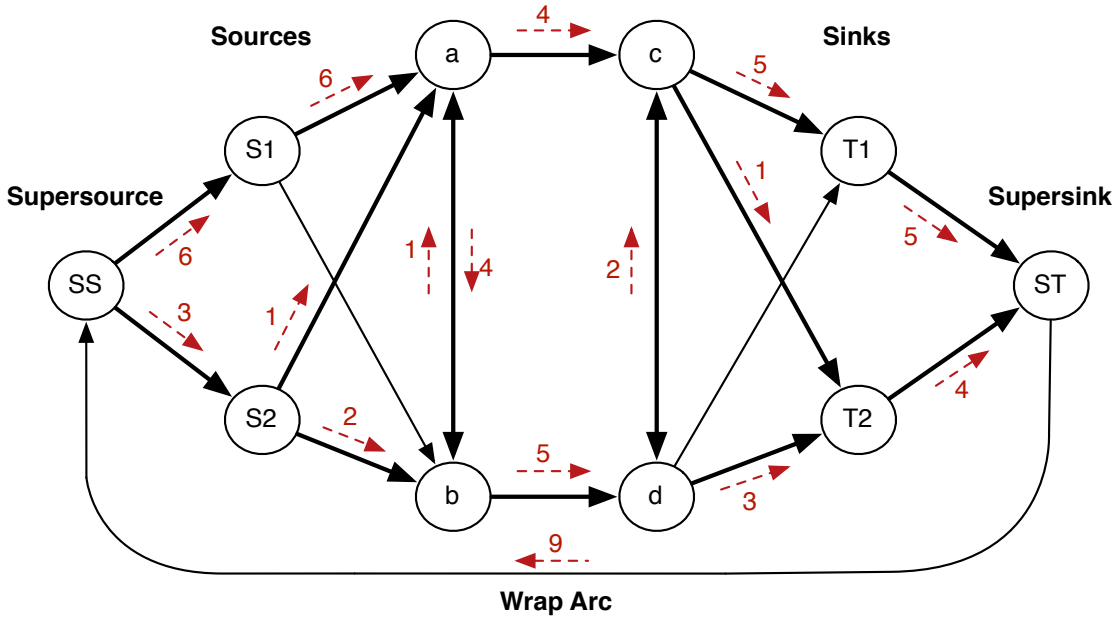

Figure 3.2: A flow network

The networks presented in Figures 3.1 and 3.2 are closely related to an aircraft fleet network if we consider a flight leg as an arc and arrival and departure locations as nodes. Hence, next we present the network structure for a fleet of aircraft and its features.

3.1.2 The aircraft fleet network structure

Here we present the aircraft fleet network structure in which all the aircraft in a fleet forms a commodity. Hence, we treat the flow network as a multi-commodity flow network. A multi-commodity flow problem is a flow problem with multiple aircraft fleet types flowing through it. With the use of this network structure events such

as 'which flight legs to fly', 'what time to fly them', and 'which aircraft fleet type to use' are decided on. The events decided on form a feasible flight schedule for a specific fleet. Figure 3.3 is an example of an aircraft fleet network structure. In this figure, the nodes are represented by small circles; the source, sink, supersource and supersink nodes are denoted by big circles; and the two types of arcs are denoted by solid and dashed arrows. Figure 3.3 also has a wrap arc which connects the supersink node and the supersource node. The flying day in Figure 3.3 starts at 6:00 and ends at 6:30. With reference to Figure 3.3, we now discuss each feature of this type of network in detail. These are:

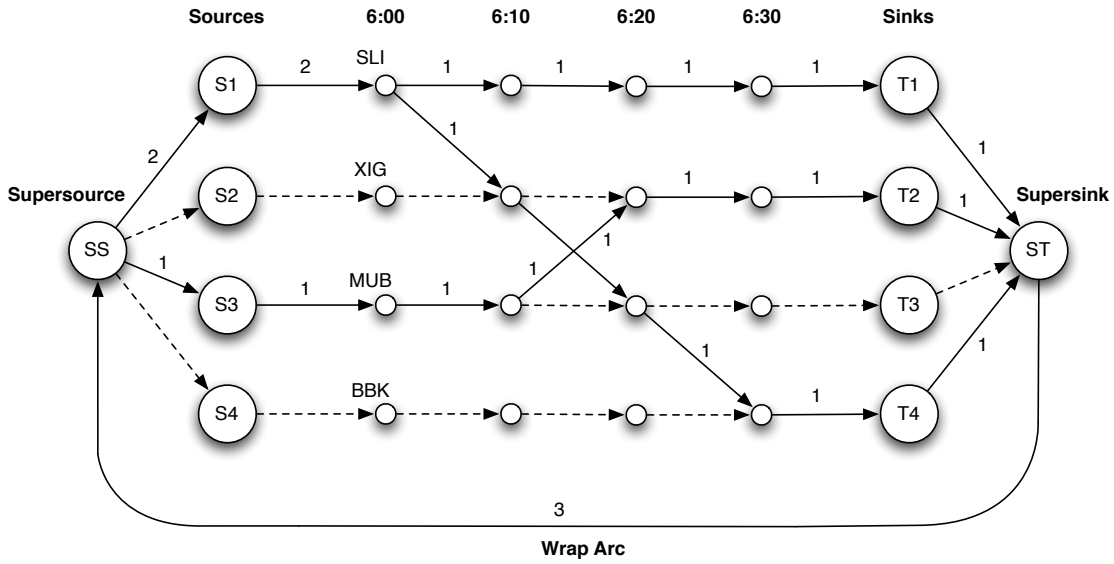

Figure 3.3: A time space aircraft network

Nodes - Nodes in this network represent a specific location at a specific time. For example SLI at 6:00 and 6:10 are two different nodes in Figure 3.3. At each node key decisions are taken by the aircraft fleet types as they travel during the day between nodes. For each fleet type there are two decisions to be taken. These decisions are 'the aircraft to load passengers and to fly them to their destinations' or 'the aircraft to remain at their current location'. There are

two types of nodes, a departure node which has a departure location and time and an arrival node which has an arrival location and time. For each departure node and arrival node chosen, there is an arc connecting them.

Arcs - Arcs connect departure nodes and arrival nodes. There are two types of arcs, namely:

Flight arc - This arc connects a departure node and an arrival node of two different locations and at two different time slots. It represents a flight leg from one location to another. For example, in Figure 3.3 the solid arrow joining the node corresponding to location SLI at 6:00 and the node corresponding to location XIG at 6:10 is a flight arc with one aircraft flowing through it.

Ground arc - A ground arc connects a departure node and an arrival node of the same location but of two consecutive time slots. It represents the case where an aircraft remains on the ground at its existing location. For example, in Figure 3.3 the arrow joining the nodes corresponding to location SLI at 6:00 and location SLI at 6:10 is a ground arc with one aircraft flowing through it. The ground arcs with zero flow are indicated with the dashed arrows.

Source ground arc - A source ground arc connects the 'supersource' node and the 'source' node of each location. For example, the source ground arc connecting the 'supersource' node (**SS**) and the 'source' node of location SLI (**S1**) has a flow of two. This represents that there are two aircraft available at location SLI at the start of the day. Similarly, the source ground arcs joining the **SS** node and the node **S3** has one aircraft flowing through it. On the other hand, there are no aircraft at nodes **S2** and **S4**. Therefore, these arcs can be used to count the number of available aircraft in the network of the k^{th} fleet type.

Sink ground arc - A sink ground arc connects the 'sink' node of each location and the 'supersink' node. This represents the number of aircraft that are staying overnight at the specific location at the end of the scheduling day. For example, the sink ground arc connecting the 'sink' node of location SLI (**T1**) and the 'supersink' node (**ST**) has a flow of 1. This represents that there is 1 aircraft that will stay overnight at location SLI. Hence, the total number of aircraft in this network is three.

Source (sink) node - The source (sink) node marks the start (end) of the network. It has the same function as the source (sink) node of the flow network discussed earlier. In Figure 3.3 there are four source (sink) nodes for each location labeled as **S1**, **S2**, **S3** and **S4** (**T1**, **T2**, **T3** and **T4**). For example, **S1** (**T1**) represents the source (sink) node of location SLI. Similarly, **S4** (**T4**) denotes the source (sink) node of location BBK.

Supersource and supersink nodes - The supersource node and supersink node are special cases of the source node and sink node, respectively. Consider the case where the network contains more than one aircraft starting location and ending location. In this case there will have to be a source (sink) node for every possible starting (ending) location. Therefore, the supersource (supersink) node will mark the start (end) of the entire network. For example, in Figure 3.3 the aircraft can start and end at SLI, XIG, MUB or BBK. Hence, there are four source and sink nodes. Therefore, a supersource node and a supersink node are needed.

Wrap arc - This is a special arc which connects the sink/supersink node to the source/supersource node. The objective of the wrap arc is to make sure that the amount of flow leaving the network is the same as the amount of flow entering the network. For example, in Figure 3.3 the wrap arc has a flow of 3 since three aircraft enters the supersource (**SS**) node and left the supersink

(**ST**) node. We use a wrap arc here because the same set of aircraft of different fleet types are used for every new flying day.

Aircraft flow - Using the aircraft in Figure 3.3, that has a starting location of MUB, we describe the aircraft flow through the network. We consider the case of location MUB. Since the aircraft has a starting location at MUB, there is a source ground arc connecting the **SS** node to the source node of location MUB (**S3**). A ground arc also joins source node **S3** to the node corresponding to location MUB at 6:00. The aircraft remained in MUB from 6:00 to 6:10. This is represented with the ground arc that joins the two nodes corresponding to MUB at 6:00 and MUB at 6:10. The flight arc connecting MUB at 6:10 to XIG at 6:20 represents a flight leg from MUB to XIG. The aircraft then remains in XIG from 6:20 to 6:30. Therefore, location XIG is the last location visited by this aircraft. Hence, there is a ground arc which connects XIG at 6:30 to the sink node of location XIG (**T2**). The sink node **T2** is then joined to the supersink (**ST**) node with a ground arc.

Conservation of flow - The flow of aircraft in and out of each node, source node, sink node, supersource node and supersink node must be conserved. For example, in Figure 3.3 there are two aircraft that have the starting location at SLI. Hence, there is a ground arc with a flow of 2 joining the source node of SLI (**S1**) to the node corresponding to SLI at 6:00. The one aircraft remained at SLI, this is represented with a ground arc joining SLI at 6:00 and SLI at 6:10. The other aircraft flies from SLI at 6:00 to XIG at 6:10. This is shown with a flight arc connecting SLI at 6:00 to XIG at 6:10. Therefore, the aircraft flow in and out of the node corresponding to SLI at 6:00 is conserved. Since two aircraft entered and two aircraft left SLI at 6:00.

Arc capacity - Each type of arc has an aircraft flow capacity. This restricts the number of aircraft flowing through an arc, these are as follows:

Flight arc capacity - The number of aircraft flowing through a flight arc must be less than or equal to one. For example, the four flight arcs in Figure 3.3 have an aircraft flow of one.

Ground arc capacity - The number of aircraft flowing through each ground arc in the k^{th} fleet's network, must be less than or equal to the number of aircraft in the fleet. For example, in Figure 3.3 there are only three aircraft in the fleet. Since there is a total of three aircraft exiting the **SS** node. Thus, the number of aircraft flowing on each ground arc cannot be greater than three.

Demand satisfaction - The aircraft flow in all the fleets' network should satisfy all the given set of flight arcs. In other words, all the required flight arcs must be flown with the use of the aircraft flow in the network.

The network presented above is used in the mathematical model formulation which we present next.

3.2 The mathematical model using the multi - commodity network flow

In this section, we only present the mathematical model of the fleet assignment problem. The goal of the model is to assign the most profitable fleet type k to a given set of flights arcs, such that the cost is minimized, whilst satisfying constraints such as the arc capacity constraint, the flow conservation constraint and the demand satisfaction constraint. For ease of explanation, we first present a basic fleet assignment model and then a fleet assignment model with 'variable time windows'. In the network formulation of the problem a flight arc of the network is a flight leg. Hence, the two models presented in the next two subsections consider a flight leg to be a flight arc. We have treated a flight leg and a flight arc as the same, hence they have

been used interchangeably. A variation will be considered later in Chapter 4, where we consider 'direct' and 'indirect' flight legs.

To create the multi-commodity aircraft fleet network structure for the models, all the airstrips in the aircraft network are duplicated into multiple copies, one for each fleet type. In each sub-network for any fleet type a time-line is associated with each airstrip, which consists of nodes that occur sequentially. The requirement that only one aircraft fleet type can be chosen to fly each flight leg jointly regulates the flows that occur in the network of each fleet type. Thus making all the networks dependent on each other. Hence, we consider the k^{th} fleet type which has its own network, namely the k^{th} network.

3.2.1 The mathematical model for the basic fleet assignment problem

Here we present the mathematical model of the fleet assignment problem. The formulation is based on the aircraft fleet network presented in Figure 3.3. We use the following notations and variables:

K : is the set of all fleet types k.

F_k : a set of flight arcs i in the k^{th} fleet's network, which carry a specific set of passengers/flight bookings. For example, there are four flight arcs in Figure 3.3.

G_k : is the set of ground arcs in the network of the k^{th} fleet type. For example, there are three ground arcs associated with location SLI in Figure 3.3.

S_k : is the number of aircraft in fleet type k.

L_k : is the set of nodes l in the network of the k^{th} fleet type.

$[t_s, t_{s+1}]$: any time interval from the s^{th} column position to the $(s+1)^{th}$ column position. For example, in Figure 3.4 the time intervals [6:00, 6:10], [6:10, 6:20], [6:20, 6:30] or [6:30, 6:40] can be used.

c_{ik} : is the cost to fly flight arc i with fleet type k, which is calculated using the operational cost of fleet type k.

x_{ik} : 1 if flight arc i is flown by fleet type k, and 0 otherwise.

y_{gk} : is the integer decision variable at the ground arc g in the network of the k^{th} fleet type. This means if the ground arc g is chosen then $y_{gk} \geq 1$ and 0 otherwise.

$b1_{lik}$: 1 if flight arc i begins at node l in the network of the k^{th} fleet type,

-1 if flight arc i ends at node l in the network of the k^{th} fleet type, and 0 otherwise.

$b2_{lgk}$: 1 if ground arc g begins at node l in the network of the k^{th} fleet type,

-1 if ground arc g ends at node l in the network of the k^{th} fleet type, and 0 otherwise.

$d1_{ik}$: 1 if flight arc i is active in the time interval $[t_s, t_{s+1}]$ in the network of the k^{th} fleet type, and 0 otherwise. For example, in Figure 3.4 *Flight arcs 1*, *2* and *3* are active in the time interval [6:10, 6:20].

$d2_{gk}$: 1 if ground arc g is active in the time interval $[t_s, t_{s+1}]$ in the network of the k^{th} fleet type, and 0 otherwise. For example, in Figure 3.4 the ground arc from SLI at 6:10 to SLI at 6:20 is active in the time interval [6:10, 6:20].

With the above definitions of variables and symbols, we now present the mathematical models for the fleet assignment problem:

$$\text{minimize} \quad \sum_{k \in K} \sum_{i \in F_k} c_{ik} x_{ik}, \tag{3.1}$$

subject to

$$\sum_{k \in K} x_{ik} = 1, \forall i \in F_k, \tag{3.2}$$

$$\sum_{i \in F_k} b1_{lik} x_{ik} + \sum_{g \in G_k} b2_{lgk} y_{gk} = 0, \forall l \in L_k, \forall k \in K, \tag{3.3}$$

$$\sum_{i \in F_k} d1_{ik} x_{ik} + \sum_{g \in G_k} d2_{gk} y_{gk} \leq S_k, \forall k \in K, \tag{3.4}$$

$$x_{ik} \in \{0, 1\}, \forall i \in F_k, \forall k \in K,$$

$$y_{gk} \geq 0, \forall g \in G_k, \forall k \in K.$$

Equation (3.1) represents the cost function. Equation (3.2) ensures that flight arc i can only be flown by one aircraft of a particular type. The amount of flow at each node is conserved with the use of Equation (3.3). The first sum calculates all the flight arcs i that depart and arrive at node l. The second sum calculates all the ground arcs g that depart and arrive at node l. Therefore, the flow is conserved by equating the sum of the departures and the arrivals at node l to zero.

Finally, Equation (3.4) counts the number of aircraft at the time interval [t_s, t_{s+1}] in the network of the k^{th} fleet type, to ensure that the amount of aircraft being used is feasible with respect to the number of aircraft in fleet type k. This is done by adding the total number of the flight arcs and ground arcs that are active in this time interval. These flight arcs include, the flight arcs that are departing at time t_s and the flight arcs that are already active in this time interval. For example, in Figure 3.4 in the time interval [6:10, 6:20] *Flight arc 3* starts at 6:10, whereas *Flight arc 1* and *2* are already flying. Both of these *Flight arcs* are regarded as active flight arcs in the time interval [6:10, 6:20]. On the other hand, *Flight arc 4* is outside of the time interval [6:10, 6:20], and therefore is not treated as active. We do not need

to consider Equation (3.4) for all possible time intervals $[t_s, t_{s+1}]$, a least activity interval is enough.

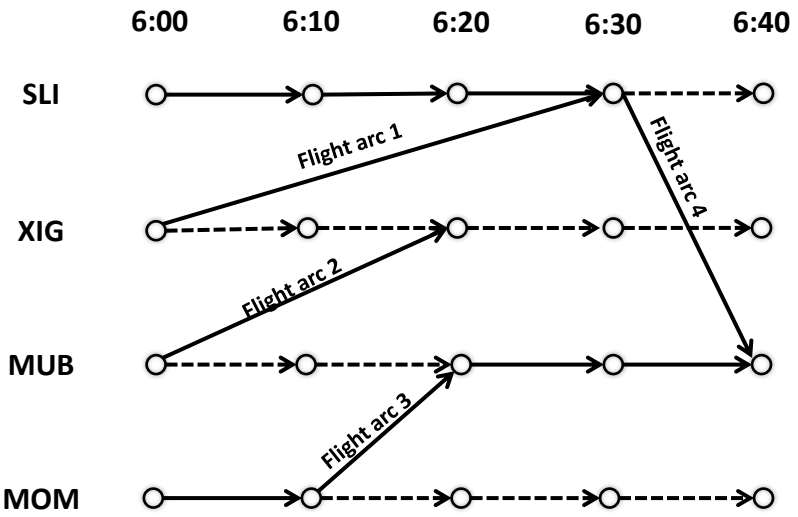

Figure 3.4: The active flight arcs and ground arcs in the time interval [6:10, 6:20]

The mathematical model presented above is an optimization model where the optimization is performed with respect to the fleet assignment. The model assumes pre-calculated flight legs with departure and arrival locations and times are given. Hence, it is a fleet assignment scheduling problem. The model optimizes the selection of which aircraft fleet type should be used for which flight leg in the schedule such that the cost is minimized.

The model presented above simplifies the fleet assignment problem. However, it does so by decreasing the opportunity of finding a better solution. A notable weakness is the requirement that each flight leg's departure time is fixed, thus reducing the flexibility of the model.

With the use of Figure 3.5, we will discuss the above weakness. In Figure 3.5 the solid arrows represent the flight arcs and are labeled with their flight arc numbers. *Flight arc 1* departs MUB at 11:50 to arrive in XIG at 12:00. *Flight arc 5* departs

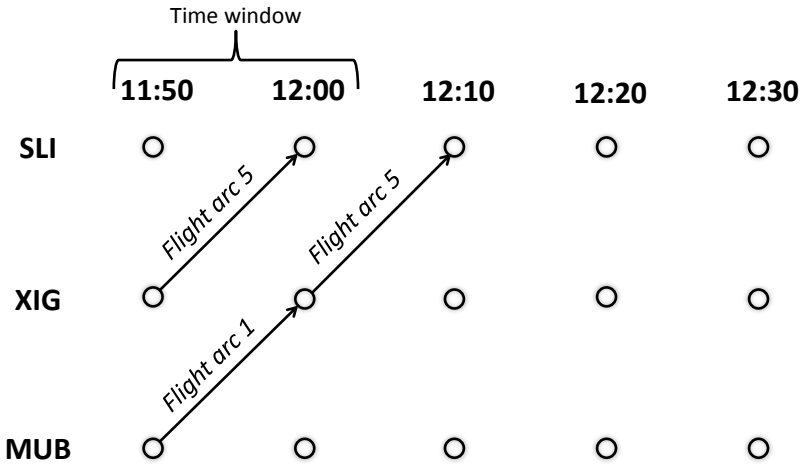

Figure 3.5: Example of the weakness of the basic fleet assignment model

XIG at 11:50 to arrive in SLI at 12:00. These two flight arcs require two separate aircraft to fly them. However, if the departure time of *Flight arc 5* is adjusted to 12:00 then these two *Flight arcs* can be flown using the same aircraft. Since *Flight arc 1* arrives in XIG at 12:00 and *Flight arc 5* then departs XIG at 12:00. This reduction of aircraft can result in significant cost savings. Therefore, a model which allows the departure time of each flight arc to be adjusted accordingly, would alleviate this deficiency. Hence, the basic fleet assignment model with time windows was introduced [14]. It is an extended version of the fleet assignment model, in that it makes use of variable time windows. Using time windows the model determines the most optimal departure time for a flight leg such that the objective function is optimized. We now present this model.

3.2.2 The fleet assignment model with time windows

In this section, we present a fleet assignment model which uses time windows. This model is similar to the previous model presented by (3.1) - (3.4), except that it now allows time windows which is represented with an additional index in the symbols and variables.

Before presenting the mathematical model based on time windows, we demonstrate the concept with an example using Figures 3.6 and 3.7. For instance, *Flight arc 2* was pre-calculated to depart MUB at 9:10 to arrive in XIG at 9:20, to satisfy the requests of the passengers on the aircraft. Suppose that the passengers' requests on the aircraft assigned to *Flight arc 2* allow the departure time to be in the interval [9:00, 9:20], resulting in the arrival time interval [9:10, 9:30]. Therefore, the departure time window for *Flight arc 2* is 9:00 to 9:20, and the respective arrival time window is 9:10 to 9:30.

Let us now consider the departure times to be 9:00, 9:10 and 9:20, and their corresponding arrival times to be 9:10, 9:20 and 9:30, respectively. To incorporate these discrete departure time intervals into the model, a flight arc is created for each departure and arrival time pair. For our example, this is done by creating three flight arcs from MUB to XIG with the following departure and arrival times: 9:00 and 9:10; 9:10 and 9:20; and 9:20 and 9:30, respectively. In Figure 3.7, we create the fleet network with the flight arcs from MUB to XIG between these time slots.

The model that we present next accommodates this with a constraint that allows only one of these flight arc copies to be selected for the solution. This allows the model to choose the most appropriate departure time for a flight arc, with respect to the other flight arcs in the network. It is important to note that using a narrow time interval will result in a more flexible model. For our example, we have used 10 minute time intervals to divide the departure time window, whereas we could have used 5 minute time intervals. This would have created a few more flight arcs copies, which gives the model a wider range of flight arc options to choose from. This however increases the size of the problem. This can be incorporated easily into the appropriate symbols and variables used in subsection 3.2.1. More specifically, the index n is added. The definitions of the symbols and the variables are as follows:

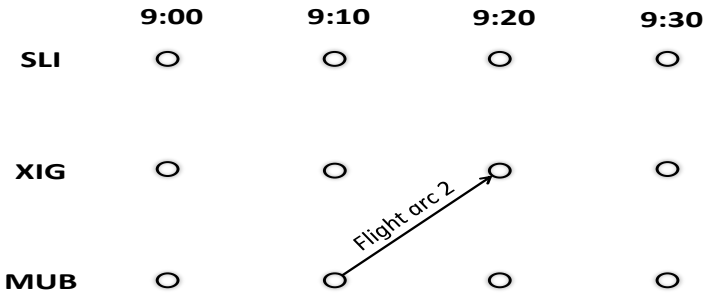

Figure 3.6: Example of a given original flight leg

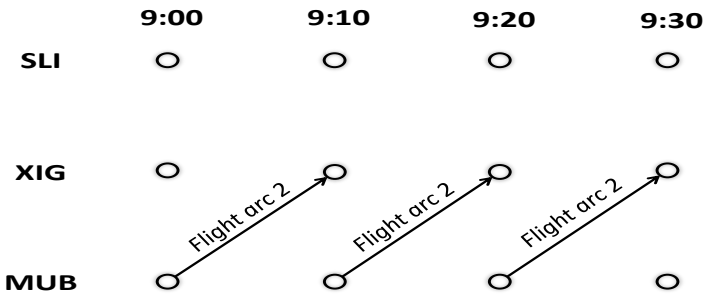

Figure 3.7: Example flight legs with time windows

K : is the set of all fleet types k.

F_k : a set of flight arcs i in the k^{th} fleet's network, which carry a specific set of passengers/flight bookings.

G_k : is the set of ground arcs in the network of the k^{th} fleet type.

S_k : is the number of aircraft in fleet type k.

L_k : is the set of nodes l in the network of the k^{th} fleet type.

$[t_s, t_{s+1}]$: any time interval from the s^{th} column position to the $(s+1)^{th}$ column position.

$b2_{lgk}$: 1 if ground arc g begins at node l in the network of the k^{th} fleet type,

-1 if ground arc g ends at node l in the network of the k^{th} fleet type, and 0 otherwise.

$d2_{gk}$: 1 if ground arc g is active in the time interval $[t_s, t_{s+1}]$ in the network of the k^{th} fleet type, and 0 otherwise.

N_{ik} : the set of flight arc copies of flight arc i in the network of the k^{th} fleet type. For example, there are three flight arc copies of *Flight arc 2* in Figure 3.7.

c_{nik} : is the cost to fly copy n of flight arc i with fleet type k, which are created using the operational costs of fleet type k.

x_{nik} : 1 if copy n of flight arc i is flown by fleet type k, and 0 otherwise.

y_{gk} : is the integer decision variable at the ground arc g in the network of the k^{th} fleet type. This means if the ground arc g is chosen then $y_{gk} \geq 1$ and 0 otherwise.

$b1_{nlik}$: 1 if copy n of flight arc i begins at node l in the network of the k^{th} fleet type,

-1 if copy n of flight arc i ends at node l in the network of the k^{th} fleet type, and 0 otherwise.

$d1_{nik}$: 1 if copy n of flight arc i is active in the time interval $[t_s, t_{s+1}]$ in the network of the k^{th} fleet type, and 0 otherwise.

With the above definitions, we now present the mathematical model for the fleet assignment problem with time windows.

$$\text{Minimize} \quad \sum_{k\in K}\sum_{i\in F_k}\sum_{n\in N_{ik}} c_{nik}x_{nik}, \tag{3.5}$$

subject to

$$\sum_{k\in K}\sum_{n\in N_{ik}} x_{nik} = 1, \forall i \in F_k, \tag{3.6}$$

$$\sum_{i\in F_k}\sum_{n\in N_{ik}} b1_{nlik}x_{nik} + \sum_{g\in G_k} b2_{lgk}y_{gk} = 0, \forall l \in L_k, \forall k \in K, \tag{3.7}$$

$$\sum_{i\in F_k}\sum_{n\in N_{ik}} d1_{nik}x_{nik} + \sum_{g\in G_k} d2_{gk}y_{gk} \leq S_k, \forall k \in K, \tag{3.8}$$

$$x_{nik} \in \{0,1\}, \forall i \in F_k, \forall k \in K, \forall n \in N_{ik},$$

$$y_{gk} \geq 0, \forall g \in G_k, \forall k \in K.$$

Equation (3.5) is the objective function. Equation (3.6) ensures that only one flight arc copy is selected from a set of flight arc copies. Equation (3.7) makes sure that the flow at each node is conserved. Equation (3.8) counts the number of aircraft on the active flight arcs and ground arcs in the time interval $[t_s, t_{s+1}]$ in the k^{th} fleet type's network. This ensures that the number of aircraft being used is feasible with regards to the number of aircraft in the k^{th} fleet.

The two mathematical models presented by (3.1) - (3.4) and (3.5) - (3.8), respectively, for the fleet assignment and the fleet assignment with time windows, assume

that all the flight legs are predetermined. The first model is a straight forward optimal fleet selection while the second model increases the flexibility of the first by incorporating time windows. These models do not consider flight leg selection optimally. For the purpose of this research, a model is developed which considers the fleet assignment using time windows, but also addresses the additional considerations such as assigning the most optimal flight legs. This model is presented in the next chapter.

Chapter 4

Fleet and flight leg assignment with time windows

In this chapter, we present the mathematical model developed to solve Sefofane's scheduling problem. We introduce a number of heuristics for the purpose of the model, in particular for the creation of the new flight legs. We describe the heuristics used to create the 'direct flight legs' and the 'indirect flight legs', and proceed to show how these are used to create an aircraft fleet network. By a direct flight leg we mean one flight leg, and by an indirect flight leg we mean a set of flight legs combined together. Small examples and diagrams will often be used to describe and motivate the use of the heuristics used. The mathematical model optimally selects both the flight legs and the fleet to fly them. The flight legs are selected from all the possible flight legs ('direct' and 'indirect') such that the daily demands in the booking list are met.

The mathematical model with time windows presented in the previous subsection improves the basic fleet assignment model (3.1) - (3.4) by using time windows. However, this model still relies on a set of predetermined flight legs. The aim of this research is to create a model which is most profitable. Hence, the selection of flight legs must be a part of the model.

The mathematical model we present here is an extension of (3.5) - (3.8) in that the decision variable incorporates the flight leg selection. In doing so, the model takes care of the passenger demands. We refer to this semi-integrated model as the fleet and flight leg assignment with time windows (FFTW) model. The FFTW model determines the best selection of flight legs to fly and the aircraft to fly them, and also determines the most appropriate departure times of each flight leg. The advantages of this type of approach is that the model is now integrated, in that it considers a large part of the flight scheduling decision process. Thus, a superior solution compared to the previous two models can be achieved. It is also in an airlines best interest to only fly flight legs which will incur less costs.

In order to fully develop this mathematical model we introduce three heuristics to demonstrate how the direct flight legs and the indirect flight legs are created. We call the three heuristics as ***Direct***, ***Indirect*** and ***Extension***, respectively. Some of these heuristics are used to reduce the size of the problem, while others are used to create indirect flight legs with which to transport passengers (tourists) to their destinations.

4.1 Heuristics to create direct and indirect flight legs

Here we present three heuristics which are used to construct direct and indirect flight legs. To create a direct/indirect flight leg with these heuristics the following flight booking conditions need to be considered:

C1. The direct/indirect flight legs created need to satisfy the flight bookings' ***Etd***, ***Ltd***, ***Eta*** and ***Lta***, also referred to as timing constraints.

C2. The total number of passengers on each flight leg of all the direct/indirect flight legs created need to satisfy at least one of the fleet type's seating capacity.

C3. An indirect flight leg can only have at most three intermediate stops, see subsection 2.1.4 for more details on the number of intermediate stops.

We begin with the first heuristic.

The *Direct* heuristic

This heuristic only creates direct flight legs by combining almost identical flight bookings. If the flight bookings cannot be joined then they are automatically treated as separate direct flight legs. We use Table 4.1, a subset of a daily booking list (see subsection 2.1.1 for details on the features of a booking list), to illustrate how flight bookings are joined to create new direct flight legs. In Table 4.1, all three flight bookings can be seen as almost identical flight bookings. Since they all have the same departure location and arrival location. These three flight bookings can therefore be combined to form the single direct flight leg MUB $\rightarrow$ SWI with booking details shown in Table 4.2, under the new group name D4. The letter 'D' is used in the ***Group Name*** in Table 4.2 to denote that the flight leg has been created by using the ***Direct*** heuristic. We also identify this direct flight leg with BI = N4. The letter 'N' is used in the booking index to denote that the flight leg is new. Observe that this resultant direct flight leg satisfies the three flight booking conditions **C1**, **C2** and **C3**. When flight bookings are joined to form a direct flight leg by this heuristic, the original flight bookings fall away. The timing constraints of the new direct flight leg, MUB $\rightarrow$ SWI, in Table 4.2 is extracted from the three flight bookings in Table 4.1. Clearly, the use of this heuristic reduces the size of the problem.

Table 4.1: The subset of a daily booking list for ***Direct***

From	*To*	*Pax*	*Group Name*	*Etd*	*Ltd*	*Eta*	*Lta*	*BI*
MUB	SWI	1	P1	6:00	11:40	6:00	15:30	1
MUB	SWI	2	P2	6:00	18:00	6:00	18:00	2
MUB	SWI	1	P3	6:00	18:00	6:00	18:00	3

Table 4.2: Resultant direct flight leg using ***Direct***

From	***To***	***Pax***	***Group Name***	***Etd***	***Ltd***	***Eta***	***Lta***	***BI***
MUB	SWI	4	D4	6:00	11:40	6:00	15:30	N4

Flight bookings cannot be joined if any of the flight booking conditions does not hold. We now consider the case where the identical flight bookings do not satisfy flight booking condition **C1**. We explain this using Table 4.3. In Table 4.3, the two flight bookings can be regarded as almost identical flight bookings. However, if we attempt to join them we can see that there is no possible set of timing constraints that would satisfy both of the flight bookings. Since the first flight booking can depart XIG no earlier than 12:00, and the second flight booking cannot depart XIG later than 11:00. This obviously violates flight booking condition **C1**. Therefore, the two flight bookings in Table 4.3 will not be joined and will be treated as two separate direct flight legs, XIG → BBK with the ***Group Name*** P5 and P6.

Table 4.3: The subset of a daily booking list where ***Direct*** fails

From	***To***	***Pax***	***Group Name***	***Etd***	***Ltd***	***Eta***	***Lta***	***BI***
XIG	BBK	1	P5	12:00	16:00	6:00	18:00	5
XIG	BBK	1	P6	8:30	11:00	6:00	16:00	6

The *Indirect* heuristic

Indirect creates indirect flight legs by joining the direct flight legs created by ***Direct***, including the direct flights legs that could not be joined by ***Direct***. This heuristic does so by creating the following sets. We use the direct flight legs in Table 4.4 to give an example of each set.

Let FL = {the set of all the direct flight legs}. For example, in Table 4.4 there are six direct flight legs, thus FL = {MUB → SWI, MUB → HND, SWI → HND, ABU → XIG, ABU → CTB, XIG → CTB}. We begin by choosing the first direct flight leg in FL i.e. MUB → SWI, and search for the following two sets:

- A = {all the direct flight legs in FL with the same departure locations as the

chosen flight leg in FL i.e. MUB → SWI}. From the set FL it is clear that A contains one such flight leg i.e. A = {MUB → HND}.

- B = {all the direct flight legs in FL with a departure location which is the same as the arrival location of the chosen flight leg in FL, and the arrival is common to a direct flight leg in A}. From the sets FL and A, it is clear that B = {SWI → HND}.

Table 4.4: The subset of a daily booking list for ***Indirect***

From	***To***	***Pax***	***Group Name***	***Etd***	***Ltd***	***Eta***	***Lta***	***BI***
MUB	SWI	1	D7	6:00	11:40	6:00	15:30	7
MUB	HND	2	D8	6:30	18:00	6:00	18:00	8
SWI	HND	1	D9	6:00	18:00	6:00	18:00	9
ABU	XIG	1	D10	6:00	13:00	6:00	15:30	10
ABU	CTB	2	D11	8:30	18:00	6:00	18:00	11
XIG	CTB	1	D12	6:00	18:00	6:00	18:00	12

The chosen flight legs MUB → SWI in FL can now be combined with the flight legs in the sets A and B in the following manner MUB → SWI → HND. The new indirect flight leg formed is given in Table 4.5. The letter 'ID' is used in the ***Group Name*** to denote that the flight leg has been created by using the ***Indirect*** heuristic. This process is then repeated for the remaining flight legs in the set FL. Hence, if the last three direct flight legs in the set FL are considered, they can be similarly linked to form the indirect flight leg ABU → CTB corresponding to BI = N14 in Table 4.5. Unlike in ***Direct***, when direct flight legs are joined to form an indirect flight leg by this heuristic, the original direct flight legs do not fall away.

We now present Figure 4.1 to show how the first indirect flight leg in Table 4.5 will be flown. This indirect flight leg consists of three direct flight legs. The path that will be flown to satisfy these three direct flight legs (see Table 4.4) is shown in Figure 4.1 with the solid arrows, (7, 8) and (8, 9). The first flight leg from MUB to SWI takes care of the direct flight leg indices {7, 8} in Table 4.4, and the second flight leg from SWI to HND takes care of direct flight leg indices {8, 9} in Table

4.4. However, in terms of the mathematical model, we represent this indirect flight leg with the flight leg from MUB to HND. This is shown with the dashed arrow. This type of flight leg is referred to as an indirect flight leg. If the mathematical model chooses the variable corresponding to the indirect flight leg MUB $\rightarrow$ HND, then this choice is based on the associated costs of the various fleet types. Since one fleet type is chosen, the underlying flight legs MUB $\rightarrow$ XIG $\rightarrow$ HND will be flown by one aircraft.

Table 4.5: Resultant indirect flight legs using ***Indirect***

From	*To*	*Pax*	*Group Name*	*Etd*	*Ltd*	*Eta*	*Lta*	*BI*
MUB	HND	4	ID13	6:30	11:40	6:00	15:30	N13
ABU	CTB	4	ID14	8:30	13:00	6:00	15:30	N14

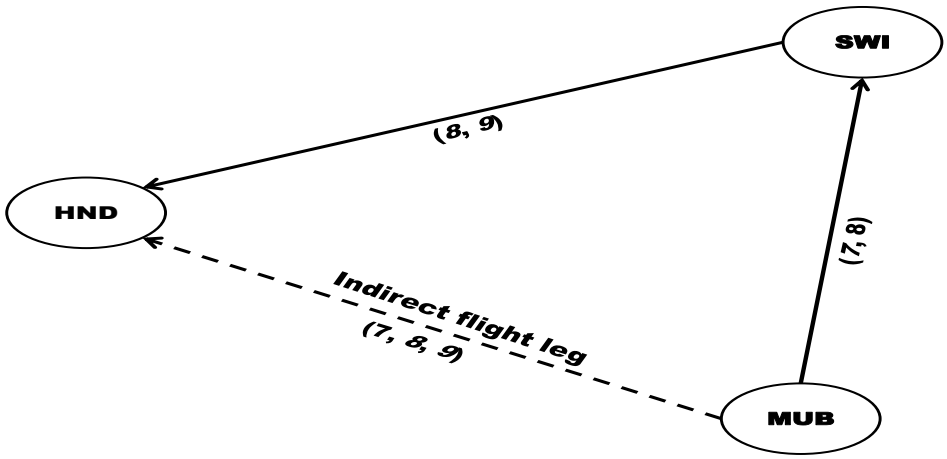

Figure 4.1: Creation of an indirect flight leg using ***Indirect***

As before, an indirect flight leg is created if the corresponding new flight legs satisfy the three flight booking conditions **C1**, **C2** and **C3**. We now consider the case in which flight booking condition **C2** is not satisfied. For instance, if the Pax of the flight booking corresponding to $BI = 7$ in Table 4.4 was 12, then the number of passengers on the indirect flight leg from MUB to SWI in Figure 4.1 would have

exceeded the seating capacity of 12 of the largest available aircraft (see subsection 2.1.6 for the aircraft specifications). Therefore, the direct flight legs in Table 4.4 would not be joined together to create the indirect flight leg in Table 4.5. Instead, all three direct flight legs would remain unchanged and will be treated as separate direct flight legs.

The *Extension* heuristic

The objective of this heuristic is to combine the indirect flight legs created by ***Indirect*** to make even longer indirect flight legs. ***Extension*** we can use two indirect flight legs, or even three indirect flight legs. We give an example where three indirect flight legs are considered. For example, the indirect flight legs corresponding to BI = N13 in Table 4.5, created using ***Indirect*** would follow the path MUB → SWI → HND. For explanation sake, we assume that two other indirect flight legs namely, MUB → BBK → SWI and SWI → XIG → HND were created using ***Indirect***. A feasible route that would satisfy all three of these indirect flight legs would be MUB → BBK → SWI → XIG → HND.

We use Figure 4.2 to describe the above process. In the upper section of Figure 4.2, we present the three indirect flight legs created by ***Indirect***. The resultant route formed, by ***Extension***, from these is shown in the lower section of Figure 4.2. As in ***Indirect***, the indirect flight legs used to create extended flight legs by ***Extension*** do not fall away. Both the extended flight legs and its component (indirect) flight legs are used to create the aircraft fleet network. The extended flight legs are also indirect, hence they will be referred to as indirect in the fleet network.

We never consider more then three indirect flight legs for this heuristic, because the resultant path would have more than three intermediate stops, which would violate flight booking condition **C3**.

The cost calculation

In the process of creating the direct, indirect and extended flight legs using each of the above heuristics the associated cost of each is calculated by considering the

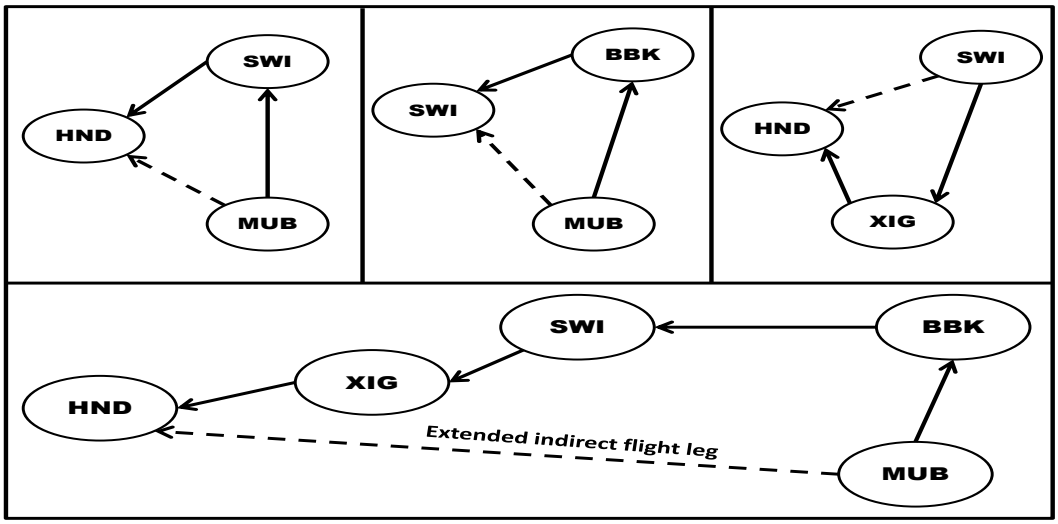

Figure 4.2: Creation of extended indirect flight leg using ***Extension***

flight legs it is made up of. For example, the cost of the indirect flight leg MUB → HND in Figure 4.1 is equal to the sum of the costs of the two flight legs MUB → SWI and SWI → HND. The cost is calculated separately for each aircraft fleet type the flight leg can be satisfied by.

4.2 The creation of the aircraft fleet network

As shown for the previous two mathematical models, in this subsection we first present the aircraft flow network using the various direct and indirect flight legs discussed in the previous subsections. In order to create an aircraft fleet network the following information of all the direct and the indirect flight legs created is required:

- The departure location and the arrival location.
- The trip time (Tt), which considers the flight time (Ft), and the take-off and landing adjustment times.
- The departure time window (DTW) and the arrival time window (ATW).

The DTW (ATW) is the time window within which the aircraft must take-off from the departure location (land at the arrival location).

We now illustrate how this information is determined with the use of an example. Again we use Table 4.6 as a subset of a daily booking list. We begin by creating the direct and the indirect flight legs.

Table 4.6: The subset of a daily booking list

From	***To***	***Pax***	***Group Name***	***Etd***	***Ltd***	***Eta***	***Lta***	***BI***
ABU	JAO	1	G1	9:00	18:00	6:00	18:00	1
HND	JAO	1	G2	6:00	18:00	6:00	15:00	2
ABU	HND	1	G3	11:00	18:00	6:00	18:00	3
ABU	XIG	1	G4	8:00	18:00	6:00	18:00	4
ABU	HND	1	G5	11:00	18:00	6:00	18:00	5
XIG	HND	1	G6	6:00	18:00	6:00	18:00	6
HND	JAO	1	G7	6:00	18:00	6:00	15:00	7
HND	MOM	1	G8	6:00	18:00	6:00	16:30	8
MOM	JAO	1	G9	6:00	18:00	6:00	15:30	9
JAO	XIG	1	G10	6:00	18:00	6:00	18:00	10
JAO	XIG	1	G11	6:00	18:00	6:00	18:00	11

4.2.1 Direct and indirect flight legs

We use ***Direct*** to join the flight bookings in Table 4.6 to form direct flight legs. There are three pairs of booking indices (BIs) in Table 4.6, each of which can be joined to create three direct flight legs. BIs 2 and 7, 3 and 5, and 10 and 11 are used to create three direct flight legs in Table 4.7. We name the booking indices associated with these flight legs as N2, N3 and N5, respectively.

Table 4.7: The direct flight legs created using ***Direct***

From	***To***	***Pax***	***Group Name***	***Etd***	***Ltd***	***Eta***	***Lta***	***BI***
HND	JAO	2	D2	6:00	18:00	6:00	15:00	N2
ABU	HND	2	D3	11:00	18:00	6:00	18:00	N3
JAO	XIG	2	D5	6:00	18:00	6:00	18:00	N5

The remaining flight bookings in Table 4.6 could not be joined by ***Direct***. Therefore, these are treated as the separate direct flight legs as shown in Table 4.8.

Table 4.8: The flight bookings in Table 4.6 that could not be joined by ***Direct***

From	***To***	***Pax***	***Group Name***	***Etd***	***Ltd***	***Eta***	***Lta***	***BI***
ABU	JAO	1	G1	9:00	18:00	6:00	18:00	1
ABU	XIG	1	G4	8:00	18:00	6:00	18:00	4
XIG	HND	1	G6	6:00	18:00	6:00	18:00	6
HND	MOM	1	G8	6:00	18:00	6:00	16:30	8
MOM	JAO	1	G9	6:00	18:00	6:00	15:30	9

The ***Indirect*** heuristic is used in Tables 4.7 and 4.8 combined to create the indirect flight legs. These are booking indices (BIs) are 1, N2 and N3 giving the indirect flight legs corresponding to the new BI = N7 in Table 4.9; the BIs N3, 4 and 6 giving the indirect flight legs corresponding to the new BI = N10 in Table 4.9; and the BIs N2, 8 and 9 giving the indirect flight legs corresponding to the new BI = N11 in Table 4.9. These flight legs are given in Figure 4.3. When the direct flight legs are joined to form the indirect flight legs, the original direct flight legs do not fall away. They are used in the fleet network.

Table 4.9: The indirect flight legs created using ***Indirect***

From	***To***	***Pax***	***Group Name***	***Etd***	***Ltd***	***Eta***	***Lta***	***BI***
ABU	JAO	5	ID7	11:00	18:00	6:00	15:00	N7
ABU	HND	4	ID10	11:00	18:00	6:00	18:00	N10
HND	JAO	4	ID11	6:00	18:00	6:00	15:00	N11

Finally, we use ***Extension*** on the indirect flight legs created by ***Indirect*** in Table 4.9 to create longer indirect flight legs. They can be chained together to form the new indirect flight leg corresponding to BI = N12 in Table 4.10 by ***Extension***. The letter 'E' is used in the ***Group Name*** to denote that the flight leg has been created by using the ***Extension*** heuristic. The final indirect flight leg's path is shown in Figure 4.4. When the indirect flight legs are joined to form the extended indirect flight legs, the original indirect flight legs do not fall away. They are used

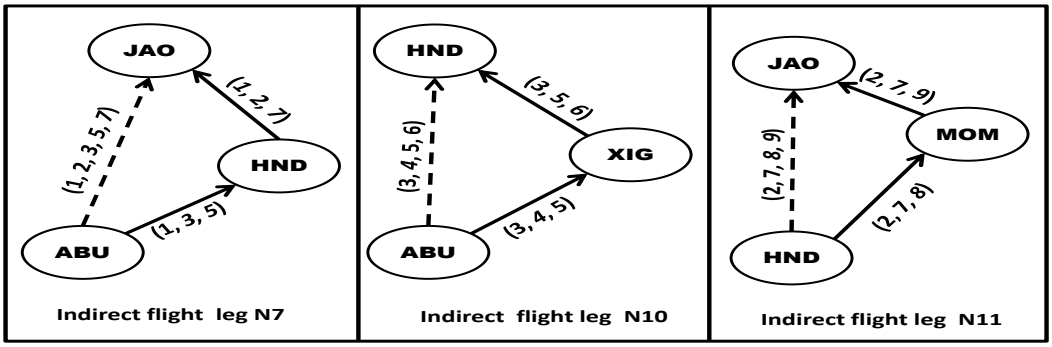

Figure 4.3: Creation of indirect flight legs using ***Indirect***

in the fleet network.

Table 4.10: The extended indirect flight leg created using ***Extension***

From	*To*	*Pax*	*Group Name*	*Etd*	*Ltd*	*Eta*	*Lta*	*BI*
ABU	JAO	9	E12	11:00	18:00	6:00	15:00	N12

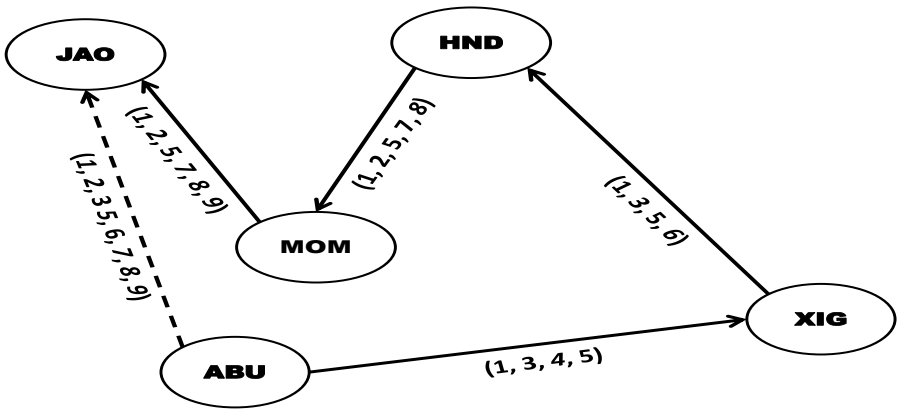

Figure 4.4: Creation of extended indirect flight leg N12 using ***Extension***

All the direct and indirect flight legs, created from the original set of flight bookings in Table 4.6 are presented in Tables 4.7 - 4.10 together with their corresponding departure and arrival locations. All these flight legs will be used to create the aircraft fleet network in subsection 4.2.4.

4.2.2 The trip time of the direct and indirect flight legs

Once the direct and the indirect flight legs are created, the next step is to calculate the trip time (Tt) of each of these direct/indirect flight legs. This is calculated using

$$Tt = 60(Ft) + Lat + Tat, \tag{4.1}$$

where the value 60 is used to convert the flight time (Ft) into minutes, and the symbols Lat and Tat are the take-off and landing adjustment times, respectively, for a specific fleet type (see Table 2.3, columns 9 and 10, respectively), and Ft is given by

$$Ft = \frac{Dt}{AS}. \tag{4.2}$$

Dt is the total distance traveled and AS is the aircraft speed for the specific aircraft fleet type being used. A detailed calculation using Equation (4.2) can be seen in subsection 2.3.1.

We will show the Tt calculation for the direct flight leg corresponding to $BI = 1$ in Table 4.8 only. The Ft of the direct flight leg corresponding to $BI = 1$ is 0.025 hr, calculated using the specifications of fleet type PC12. This is based on the distance $dist(ABU, JAO) = 11$ km and the aircraft speed $AS = 440$ km/hr. Therefore, the Tt is given by:

$$\begin{aligned} Tt &= 60(Ft) + Lat + Tat \\ &= 60(0.025\ hr) + 12\ min + 12\ min, \\ &= 25.5\ hr, \end{aligned}$$

where Lat and Tat denote the take-off and landing adjustment time, respectively, of aircraft fleet type PC12, and the values of which have been taken from Table 2.3. The Tt for the other direct and indirect flight legs in Tables 4.7 - 4.10 can similarly be calculated.

The Ft for each indirect flight is calculated according to the flight legs that it is composed of. The calculated Tt values for all the direct and indirect flight legs in

Tables 4.7 - 4.10 are given in Table 4.11. All these Tt values were calculated using data from one fleet type only. The Tt using the specifications of the other fleet types can be calculated similarly.

The BI values given in the first column of Table 4.11 correspond to the BI values of all the new direct and indirect flight legs given in Tables 4.7 - 4.10. The flight booking indices given in the last column of Table 4.11 are the original BIs of the flight bookings in Table 4.6. The *Flight legs* in column 2 of Table 4.11 satisfy the corresponding flight booking in the last column. For example, the flight leg corresponding to BI = N2 in column 1, Table 4.11, satisfies the flight bookings (2 and 7 in the last column, Table 4.11 corresponding to $BI = 2$ and 7 in Table 4.6). In column 2 of Table 4.11, we have given the first departure and last arrival location of the direct/indirect flight legs. Below each indirect flight leg in Table 4.11, we have presented the component flight legs in brackets. For example, for the indirect flight leg ABU $\rightarrow$ JAO corresponding to BI = N7, the component flight legs ABU $\rightarrow$ HND $\rightarrow$ JAO are given underneath ABU $\rightarrow$ JAO. In addition, we have presented the total Pax of the direct and indirect flight legs in column 3. Furthermore, we have given the Pax on each flight leg in the indirect flight legs below in brackets.

4.2.3 The departure and arrival time windows

To calculate the DTW and the ATW, we consider the following specifications of each direct/indirect flight leg:

- The departure timing constraint, $[Etd, Ltd]$.
- The arrival timing constraint, $[Eta, Lta]$.
- The trip time (Tt).

Given the trip time, Tt, between the departure and arrival location, incurred by a fleet, and the timing constraints $[Etd, Ltd]$ and $[Eta, Lta]$, we use the following procedure to calculate the DTW and the ATW of the flight legs.

Table 4.11: Additional information of the flight legs in Tables 4.7 - 4.10

BI	***Flight legs***	***Pax***	***Tt*** min	***DTW***	***ATW***	***Flight bookings***
1	ABU → JAO	1	25.5	[9:00, 17:30]	[9:30, 18:00]	1
N2	HND → JAO	2	25.9	[6:00, 14:30]	[6:30, 15:00]	2, 7
N3	ABU → HND	2	26.2	[11:00, 17:30]	[11:30, 18:00]	3, 5
4	ABU → XIG	1	26.9	[8:00, 17:30]	[8:30, 18:00]	4
N5	JAO → XIG	2	26.5	[6:00, 17:30]	[6:30, 18:00]	10, 11
6	XIG → HND	1	28.4	[6:00, 17:30]	[6:30, 18:00]	6
N7	ABU → JAO (ABU → HND → JAO)	5 (3 → 3)	52.1	[11:00, 14:00]	[12:00, 15:00]	1, 2, 3, 5, 7
8	HND → MOM	1	28.9	[6:00, 16:00]	[6:30, 16:30]	8
9	MOM → JAO	1	27.4	[6:00, 15:00]	[6:30, 15:30]	9
N10	ABU → HND (ABU → XIG → HND)	4 (3 → 3)	55.2	[11:00, 17:00]	[12:00, 18:00]	3, 4, 5, 6
N11	HND → JAO (HND → MOM → JAO)	4 (3 → 3)	56.3	[6:00, 14:00]	[7:00, 15:00]	2, 8, 7, 9
N12	ABU → JAO (ABU → XIG → HND → MOM → JAO)	9 (4 → 4 → 5 → 6)	111.6	[11:00, 13:00]	[13:00, 15:00]	1, 2, 3, 4, 5, 6, 7, 8, 9

BI: Booking index; ***Pax***: Number of passengers; ***Tt***: Trip time;

DTW: Departure time window; ; ***ATW***: Arrival time window.

We use d_0, d_1, ..., $d_n \in [Etd, Ltd]$, $d_k < d_{k+1}$, and calculate the smallest d_i and the largest d_j for which

$$d_i + ceil(Tt), d_j + ceil(Tt) \in [Eta, Lta]. \tag{4.3}$$

where d_i, d_j is a time in the interval $[Etd, Ltd]$ and *ceil* is a function that rounds a number to the nearest integers greater than or equal to itself. We then assign the $DTW = [d_i, d_j]$ and the $ATW = [d_i + ceil(Tt), d_j + ceil(Tt)]$.

For example, the indirect flight leg corresponding to BI = N7 in Table 4.9, has the following specifications:

- $[Etd, Ltd]$ = [11:00, 18:00].
- $[Eta, Lta]$ = [6:00, 15:00].
- $Tt = 52.1\ min.$

The smallest (largest) d_i (d_j)$\in [Etd, Ltd]$ that satisfies the expression in Equation (4.3) is 11:00 (14:00). If we choose $d_n >$ 14:00 then the resultant arrival time will not be in the interval [6:00, 15:00]. Therefore, the DTW = [11:00, 14:00] and the corresponding ATW = [12:00, 15:00]. The DTW and ATW of the other direct/indirect flight legs in Tables 4.7 - 4.10, can be calculated similarly. These are shown in Table 4.11.

4.2.4 The aircraft fleet network

Now that all the required information has been calculated, we can create the flight arcs which form the aircraft fleet network. We create the aircraft fleet network for all the direct and indirect flight legs in Table 4.11. We have presented the additional information of each direct/indirect flight leg inTable 4.11, namely, the trip time (Tt), DTW, ATW, and departure and arrival locations. Using the above information we have created the aircraft fleet network in Figure 4.5 (corresponding to the flight

bookings in Table 4.6). As before we denote the flight arcs corresponding to the direct (indirect) flight legs by the solid (dashed) arrows in Figure 4.5.

The network presented in Figure 4.5, however, does not include flight arc copies which are essential for the mathematical model that follows next. We now discuss how the flight arc copies for the direct and indirect flight legs in Table 4.11 are created.

To describe the flight arc copies we use the indirect flight leg corresponding to BI = N12 in Table 4.10. This flight leg has the following information (taken from the last row in Table 4.11), e.g. $Tt = 111.6\ min$, DTW = [11:00, 13:00], ATW = [13:00, 15:00], the departure location ABU, and the arrival location JAO.

We now use the above information to show the possible flight arc copies of N12. From the above information we therefore create flight arcs copies between location ABU and location JAO between the departure and arrival time pairs: 11:00 and 13:00; 11:10 and 13:10; 11:20 and 13:20; ... ; and 13:00 and 15:00, respectively. To draw a network with all the flight arc copies would complicate the presentation of Figure 4.5, thus we have only shown the flight arc from the node corresponding to ABU at 11:00 and the node corresponding to JAO at 13:00, labeled with the flight arc number N12. The flight arc copies for the rest of the direct and indirect flight legs can be created similarly.

Ground arcs in the network in Figure 4.5 have been implicitly represented. For example, if the node ABU at 11:00 had three aircraft flowing into it and flight arc N12 was chosen for the final solution, then one aircraft would flow through flight arc N12 and the other two aircraft would flow through a ground arc that connects ABU at 11:00 and ABU at 11:30.

Next, we show how the aircraft flow network in Figure 4.5 is embedded into a mathematical model, formulated to optimize the selection of fleet and flight legs. The mathematical model that we present next considers the flight arc copies of the network in Figure 4.5, as described above.

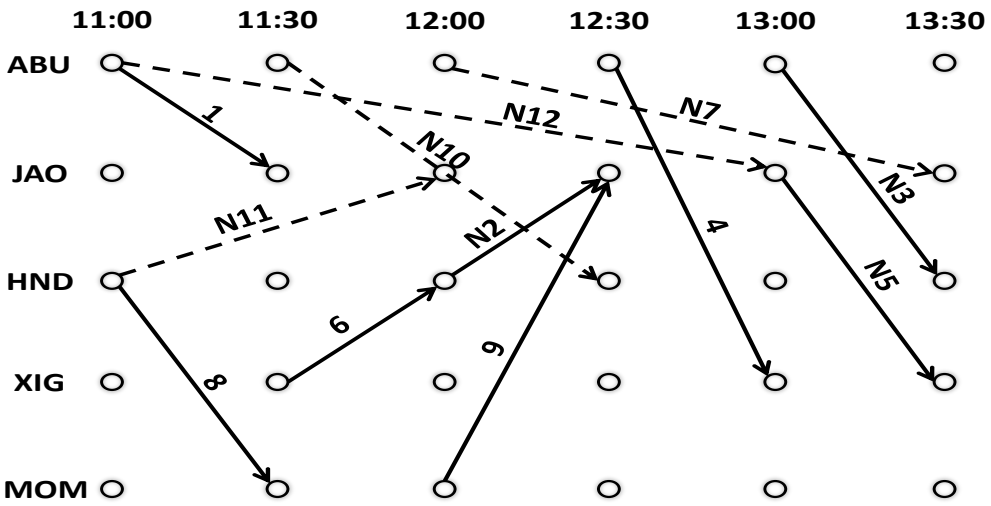

Figure 4.5: The aircraft flow network for the flight legs in Table 4.11

4.3 The mathematical model

Now that we have shown how the aircraft flow network is created. We will present the mathematical model which uses this flow network to assign the most optimal fleet and flight legs.

To ensure that all the flight arcs chosen for the final solution only contains each flight booking (the flight bookings in Table 4.6) once, a record of the flight arcs satisfying each flight booking is created in Table 4.12. The first column in Table 4.12 contains the original flight booking indices of Table 4.6, and the second column contains the name of the direct and indirect flight legs that cater for the corresponding *BI* in column 1. Looking at Table 4.11, one can see that there are flight bookings that are satisfied by more than one flight arc. For example, flight booking 1 is satisfied by flight arc 1, N7 and N12, see Table 4.12, row 1. However, the model ensures that only one of these flight arcs are chosen for flight booking 1 in the final solution. Therefore a solution to the problem in Table 4.12 must only have one flight leg that satisfies each booking in Table 4.6. For example, flight leg

1 and flight leg N12 cannot be a part of any solution since they both satisfy flight booking 1. This means only one flight arc in the set {1, N7, N12} can be chosen to cater for flight booking 1 in Table 4.6 (also given in Table 4.12, column 1).

Table 4.12: Record of flight arcs

BI	***Flight arcs***
1	1, N7, N12
2	N2, N7, N11, N12
3	N3, N7, N10, N12
4	4, N10, N12
5	N3, N7, N10, N12
6	6, N10, N12
7	N2, N7, N11, N12
8	8, N11, N12
9	9, N11, N12
10	N5
11	N5

Our objective is to cater for all the flight bookings in column 1, Table 4.12. One can see that there are many possible feasible solutions that cater for the entire booking list in column 1. For example, the flight arcs N5 (caters for bookings 10 and 11) and N12 (caters for bookings 1, 2, 3, 4, 5, 6, 7, 8 and 9) constitute a feasible solution which satisfies all the flight bookings in Table 4.12 (or Table 4.6) only once. Similarly, the flight arcs 4, N5 (caters for 10 and 11), 6, N7 (caters for 1, 2, 3, 5 and 7), 8 and 9 constitute a feasible solution. Clearly, for each booking in each row in Table 4.12 only one flight arc in the row has to be chosen. Hence, we assign a variable for each flight arc in Table 4.12. For the particular example we consider 12 variables in Table 4.12. Each row in Table 4.12 will be represented by a constraint in the mathematical formulation (see Equation (4.5)). The binary variable associated with the i^{th} flight arc in Table 4.12 is denoted as x_{nik}. Hence, there are three binary variables, x_{n1k}, x_{n7k} and x_{n12k}, associated with row 1 in Table 4.12, and one of these will take the value 1 (the binary variables x_{n1k}, x_{n7k} and x_{n12k} correspond to arcs 1, N7 and N12, respectively). With the use of these variables the FFTW model creates

longer routes by joining the most optimal flight arcs to each other.

The selection of these flight arcs can easily be included into the appropriate symbols and variable used in subsection 3.2.2, by introducing a new index b. The definitions of the symbols and variables are as follows:

K : is the set of all fleet types k.

B : the set of flight bookings b in the booking list. For example, the flight bookings in Table 4.6.

F : the set of flight arcs f, which includes the flight arcs that satisfy flight bookings and the flight arcs that do not satisfy any flight bookings i.e. the empty flights.

F_{bk} : the set of flight arcs $i \in F$ satisfying flight booking b (e.g. the flight arcs in column 1, Table 4.11) in the network of the k^{th} fleet type.

E_k : the set of flight arcs $e \in F$ satisfying no flight booking i.e. the empty flights in the network of the k^{th} fleet type.

G_k : is the set of ground arcs g in the network of the k^{th} fleet type.

S_k : is the number of aircraft in fleet type k.

L_k : is the set of nodes l in the network of the k^{th} fleet type.

R_k : is the set of source ground arcs α that join the source node to the first node in the network of the k^{th} fleet type (see subsection 3.1.2).

$b2_{lgk}$: 1 if ground arc g begins at node l in the network of the k^{th} fleet type,

-1 if ground arc g ends at node l in the network of the k^{th} fleet type, and 0 otherwise.

N_{bik} : the set of flight arc copies of flight arc $i \in F_{bk}$ satisfying flight booking b in the network of the k^{th} fleet type.

N_{ek} : the set of flight arc copies of flight arc $e \in E_k$ satisfying no flight booking in the network of the k^{th} fleet type.

c_{nik} : is the cost to fly copy n of the flight arc copies of flight arc $i \in F_{bk}$ satisfying flight booking b in the network of the k^{th} fleet type.

c_{nek} : is the cost to fly copy n of the flight arc copies of flight arc $e \in E_k$ satisfying no flight booking in the network of the k^{th} fleet type.

x_{nik} : 1 if copy n of the flight arc copies of flight arc $i \in F_{bk}$ satisfying flight booking b is flown by fleet type k, and 0 otherwise.

x_{nek} : 1 if copy n of the flight arc copies of flight arc $e \in E_k$ satisfying no flight booking is flown by fleet type k, and 0 otherwise.

y_{gk} : is the integer decision variable at the ground arc g in the network of the k^{th} fleet type. This means if the ground arc g is chosen then $y_{gk} \geq 1$ and 0 otherwise.

$z_{\alpha k}$: 1 if source ground arc α is flown by fleet type k, and 0 otherwise. Unlike the description of the source ground node presented in subsection 3.1.2, here we take the provision that if there are more than one aircraft at a certain location then we consider a source ground arc for each.

$b1_{nlik}$: 1 if copy n of the flight arc copies of flight arc $i \in F_{bk}$ satisfying flight booking b begins at node l in the network of the k^{th} fleet type,

-1 if copy n of the flight arc copies of flight arc $i \in F_{bk}$ satisfying flight booking b ends at node l in the network of the k^{th} fleet type, and 0 otherwise.

$b3_{nl\alpha k}$: -1 if source ground arc α ends at node l in the network of the k^{th} fleet type, and 0 otherwise.

$b4_{nlek}$: 1 if copy n of the flight arc copies of flight arc $e \in E_k$ satisfying no flight booking begins at node l in the network of the k^{th} fleet type,

-1 if copy n of the flight arc copies of flight arc $e \in E_k$ satisfying no flight booking ends at node l in the network of the k^{th} fleet type, and 0 otherwise.

Using the above definitions, we now present the mathematical model for the FFTW model.

$$\text{Minimize} \quad \sum_{k\in K}\sum_{i\in F_{bk}}\sum_{n\in N_{bik}} c_{nik}x_{nik} + \sum_{k\in K}\sum_{e\in E_k}\sum_{n\in N_{ek}} c_{nek}x_{nek}, \tag{4.4}$$

subject to

$$\sum_{k\in K}\sum_{i\in F_{bk}}\sum_{n\in N_{bik}} x_{nik} = 1, \forall b \in B, \tag{4.5}$$

$$\sum_{i\in F_{bk}}\sum_{n\in N_{bik}} b1_{nlik}x_{nik} + \sum_{g\in G_k} b2_{lgk}y_{gk} + \sum_{\alpha\in R_k} b3_{l\alpha k}z_{\alpha k} + \sum_{e\in E_k}\sum_{n\in N_{ek}} b4_{nlek}x_{nek} = 0, \tag{4.6}$$

$$\forall b \in B, \forall l \in L_k, \forall k \in K,$$

$$\sum_{\alpha\in R_k} z_{\alpha k} \leq S_k, \forall k \in K, \tag{4.7}$$

$$x_{nik} \in \{0,1\}, \forall b \in B, \forall i \in F_{bk}, \forall k \in K, \forall n \in N_{bik},$$

$$x_{nek} \in \{0,1\}, \forall e \in E_k, \forall k \in K, \forall n \in N_{ek},$$

$$y_{gk} \geq 0, \forall g \in G_k, \forall k \in K.$$

$$z_{\alpha k} \in \{0,1\}, \forall k \in K, \forall \alpha \in R_k.$$

Equation (4.4) is the objective function. The first sum calculates the cost of the flight arcs $i \in F_{bk}$ satisfying flight booking b, and the second sum is the cost due to

empty flight arcs. Equation (4.5) ensures that only one flight arc copy that satisfies booking b in the booking list is selected from a set of flight arc copies. Equation (4.6) makes sure that the flow at node l is conserved. The first sum calculates all the flight arcs $i \in F_{bk}$ that satisfy flight booking b that depart and arrive at node l. The second sum calculates all the ground arcs g that depart and arrive at node l. The third sum calculates all the source ground arcs α that arrive at node l. The forth sum calculates all the empty flight arcs $e \in E_k$ that depart and arrive at node l. Therefore, the flow is conserved by equating the sum of the departures and the arrivals at node l to zero. Equation (4.7), allows the model to use the source ground arcs that are available i.e. have aircraft on them, to start the initial flow in the network. This equation also ensures that the number of aircraft used in the entire network is feasible with regards to the total number of available aircraft at each k^{th} fleet.

The objective of the model is to choose optimally a set of flight arcs which satisfies all the given flight bookings only once, whilst assigning the most optimal fleet type. For example, given the flight bookings in Table 4.6, the objective is to fly all of them by choosing specific flight arcs in Table 4.11.

Chapter 5

Computational results

In this chapter, we present and discuss the numerical implementation of the final mathematical model presented in Chapter 4. The intention is to solve the model developed and analyze the results obtained. The results of our model are then compared with a manually created flight schedule. Our numerical experiments were carried out using three fleet types. These are the C206 consisting of six aircraft, the C208 consisting of five aircraft, and the PC12 consisting of one aircraft. The specifications for all these aircraft are given in Table 2.3.

5.1 Equipment and software specifications

In this section, we present the specifications of the computer used. Furthermore, we describe the different stages of the implementation process and the software used.

The optimization model was solved using the CPLEX solver, version 12.1, under Linux environment. The specifications of the computer used are presented in Table 5.1. The CPLEX solver was chosen for its relative performance against the SCIP 1.2.0 solver. It was found that the optimal solution was obtained relatively faster. A multistage approach was required to solve the problem, which we describe next.

Stage 1

Table 5.1: Computer specifications

Computer brand	HP
Model	HP Pavilion Dv6
Processor	Intel(R) Core(TM) i3 CPU
Processor speed	1
Number of processors	2.27 GHz
Number of cores	4
RAM size	4 GB
Operating system	Windows 7 Professional
System type	32 bit Operating system

The first stage of the solution process involves a program, which was written in Matlab, to create all the direct and indirect flight legs based on the heuristics ***Direct***, ***Indirect*** and ***Extension*** in Chapter 4. The inputs used in the program are airstrip location data, booking information and aircraft specifications. These direct and indirect flight legs are then used by the program to create an aircraft fleet network. The Matlab program also converts the features of the time-space network (e.g. Figure 4.5) into the required input format for the executable Zuse Institute Mathematical Programming Language (ZIMPL) [13] file (a full listing of the Matlab program has been appended at the end of Appendix XII). A sample input format file of the ZIMPL model is given in Appendix III, and will be referred to in the descriptions presented below. The input format of the ZIMPL model consists of the following items [13].

Sets : A set is defined, for a number of program variables. Each component in the set is a value the specific program variable can take on. Sets are defined with the following statement:

$$\textbf{set}\ setname := \{component_1, \ldots, component_n\}$$

where,

setname - is the name given to the set.

$component_i$ - is the i^{th} component, $i = 1 \ldots n$.

For example, set $VG := \{1 \text{ to } 4602\}$ in Appendix III is the set of ground arcs, $i = 1, 2, \ldots, 4602$.

Parameters : A parameter can be defined by the following statement:

parameter $parname[set_1 * \ldots * set_n] := \{< t_1 > v_1, \ldots, < t_n > v_n\}$

where,

$parname$ - is the name given to the parameter.

set_j - is the name of the set that forms the tuple that will be used in the parameter with $j = 1 \ldots n$.

t_j - is the tuple $c_1, \ldots, c_n$, where $c_j \epsilon\ set_j$ formed by the set $set_1 * \ldots * set_n$, yielding the value v_j, $j = 1 \ldots n$.

v_j - is the value the parameter will take on for tuple t_j with $j = 1 \ldots n$.

We consider two examples from Appendix III to explain the above format. First we consider the cost of flight arcs. Here c is the cost parameter for each flight arc and V is the set of flight arcs, see Appendix III. The values of c are read from the file "cost.dat". These values have to be presented in the format: <1n> 2n, where 1n is the index and 2n is the cost value. For example, if the set $V = \{1, 2, 3\}$ i.e. there are three flight arcs then param $c[V]$ will be defined as $\{< 1 > 20, < 2 > 35, < 3 > 15\}$, where $< 1 > 20$, for example, indicates that the cost of flight arc 1 is 20.

For the second example, we use the fleet type. In this case air is the parameter for the number of aircraft in each fleet type and A is the set of fleet types. The value of air are read from the file "fleets.dat". These values have the following format: <1n> 2n, where 1n is the index of the fleet type and 2n is the number

of aircraft in the fleet type value. Therefore, if the set $A = \{1, 2, 3\}$ i.e. there are three fleet types then param $air[A]$ will be defined as $\{< 1 > 5, < 2 > 6, < 3 > 1\}$, where $< 1 > 5$, for example, indicates that there are five aircraft in fleet type 1.

Variables : This defines all the variables. A variable can be defined in the following way:

var *variablename type*

where,

variablename - is the name of the variable.

type - is the type of variable namely: continuous (real); binary; or integer depending on the variable type needed.

For example, var $x[V]$ in Appendix III represents the binary variable x for all the flight arcs in set V.

Objective : This statement defines the objective function. The objective function is defined as follows:

minimize/maximize $objname : objfun$

where,

objname - is the name of the objective function.

$objfun$ - is the right hand side of the objective function.

Constraints : These define all the constraints in the model using the following format:

subto $conname : confun$

where,

conname - is the name of the constraint.

$confun$ - is the constraint.

The above information will be used by the ZIMPL executable file in the next stage.

Stage 2

In order for the CPLEX to read the model, the ZIMPL model first needs to be translated into a linear programming model expressed in lp format [12] using the ZIMPL executable file. The lp format is a required format for CPLEX. This is equivalent to the format of a linear programming problem in the following form:

$$\min \quad C^T \underline{x}, \tag{5.1}$$

subject to

$$A\underline{x} \leq b. \tag{5.2}$$

The lp file created is read into the CPLEX solver, and the model is solved.

There is always a feasible solution to the problem considered except for the case where there are not enough aircraft available. In this case, additional aircraft will be added manually.

Once the solver has produced a solution, the final stage is to generate a human readable flight schedule from the solution. We have generated five flight schedules each for different sets of flight bookings consisting of 25, 50, 75, 100, 125, 150 and 175 bookings. These seven sets of flight booking lists are created in increments of 25 flight bookings. That is for a given set of 175 flight bookings, the first set consists of 1 - 25 flight bookings, the second set consists of 1 - 50 flight bookings and so on. Each booking list and its optimal flight schedule (human readable flight schedule) are presented in Appendices IV - X. Each of these human readable flight schedules are self-explanatory except for the rows containing $Pax = 0$ and $BI =$ EMPTY.

These are the flight legs that are flown empty to another location where there are awaiting tourists. In Table 5.2, we summarize features of each schedule such as the number of flight bookings, the number of aircraft used (NAU), the number of flight legs (NFL), the CPU time (combined CPU for CPLEX and the Matlab program) in seconds (sec), the flying costs (FC) and the load factor (LF). The LF is given by

$$LF = \frac{NP}{NAS}, \tag{5.3}$$

where NP is the number of passengers on the flight legs used in the solution and NAS is the number of seats available on the aircraft for these flight legs. The LF gives the airline a measure of the number occupied seats on the flight legs. $LF = 1$ indicates that every seat in the aircraft has a passenger allocated to it.

Table 5.2: The summarised results for the sets of flight bookings used

Number of flight bookings	***NAU***	***NFL***	***CPU sec***	***LF***	***FC $***
25	6	25	339.68	0.3	7497
50	8	41	457.45	0.4	13056
75	9	48	626.15	0.5	14773
100	9	53	889.67	0.5	16065
125	10	67	925.21	0.4	20839
150	12	77	1033	0.5	22315
175	12	72	1140.17	0.5	23130

NAU: Number of aircraft used; ***NFL***: Number of flight legs; ***LF***: Load factor; ***FC***: Flying costs.

It can be seen from Table 5.2 that the calculated load factors are very low. There are few factors contributing to this low ratio. Looking at the geographical spread of the safari camps Figure 2.1, we see that there are very concentrated areas but also quite a few scattered areas. The load factors for the concentrated area are normally very high. However, the load factors for the scattered areas and unpopular camps are reasonably low as there are usually only one or two tourists flying there. In addition, some tourists have ridiculous timing constraints which often requires an aircraft to fly there solely for them, affecting the load factors.

Next, we make a detailed comparison of one optimal flight schedule predicted by the model with that of a manual schedule provided by Sefofane.

5.2 Manual schedule vs model schedule

In this section, we present a detailed comparison between a manual schedule provided by Sefofane and the schedule produced by us.

Sefofane provided us with a real life instance of a daily booking list (see Appendix XI). Flight bookings for a typically busy day was given for testing. It consists of 175 flight bookings with over 300 passengers in total. Sefofane also provided us with their best known manual flight schedule that would cater for these flight bookings.

The summary of the comparison between the manual schedule and the schedule obtained by us is presented in Table 5.3. The key criteria used in evaluating the performance of our model is the total flying cost. It can be seen from the table that our model has obtained an improved flight schedule, which is much cheaper than the manual schedule. A cost saving of $2602.44 i.e. 10.11% improvement was achieved, for the booking list of 175 bookings. This is due to the fact that the final solution has used 72 flight legs, as opposed to the manual schedule which has used 83 flight legs. Using fewer flight legs could also reduce the maintenance cost of the aircraft, as the aircraft are accumulating less flying hours per day. The cost presented in Table 5.3 is a daily cost. The savings amounts to a total monthly cost saving of $78073.20, which is quite significant for a small company like Sefofane.

To perform the daily scheduling of about 175 flight bookings, Sefofane has to employ three full time professional schedulers. A normal scheduling day begins at 8:00 and ends at 17:00. During this time, the schedulers create a flight schedule for the following operating day. It can be seen in Table 5.3 that our model produces the flight schedule in 19 minutes as opposed to the manual schedule which requires 8 hours work of three employees. Clearly, the company can make significant saving by

either hiring lower number of schedulers or assigning the schedulers to other duties.

The load factor for our schedule is equal to the manual schedule, even though we expected it to be higher. This is due to the fact that the load factor is considered by the manual schedulers whereas our model does not explicitly include the load factor.

Table 5.3: Manual schedule vs our schedule

	Manual schedule	**Our schedule**
Flying cost $	25732.44	23130
Processing time *sec*	28800	1140.17
Number of flight legs	83	72
Average load factor	0.5	0.5
Number of C206 flight legs	41	15
Number of C208 flight legs	38	47
Number of PC12 flight legs	4	10

Another noted difference between the two schedules is the number of flight legs flown by each fleet type. It can be seen in Table 5.3 that the manual schedule makes more use of the fleet type C206, whereas our schedule uses more of the fleet type C208. The C206 is the cheapest aircraft, however it can only seat 5 passengers. The C208 is more expensive with regards to flying costs but it can seat up to 12 passengers. Passengers also prefer the C208 as it is more comfortable and luxurious.

Chapter 6

Conclusion

The objective of this research was to develop a semi-integrated model to solve both the flight leg selection and the fleet assignment problem. The semi-integrated model, adopted here using a time-space network, has not been reported in the literature.

To develop the semi-integrated model, we first introduced three heuristics namely, ***Direct***, ***Indirect*** and ***Extension***, to create direct and indirect flight legs. We used these direct and indirect flight legs to construct an aircraft fleet network structure.

To evaluate the effectiveness of our model, we compared the schedule of 175 flight bookings to that of a manual schedule provided by Sefofane. It was found that our solution achieved a daily cost saving of 10%, compared to the cost of the manual schedule.

The benefits of such a saving for a small airline like Sefofane is quite significant. The on-demand flight scheduling problem considered here has, as with many other on-demand scheduling problems, a number of challenging features. The daily variability in the demand scenario is a challenging issue of the problem. The other challenging issue is that there are many small group of tourists with varied demand scenarios, e.g. timing constraints. Under these circumstances, the introduction of three heuristics has made the problem manageable. These heuristics have reduced the size of the problem significantly. The fact that the model was solved for the

booking list of size 175 within 19 minutes clearly justifies the use of the heuristics.

Although the model was unable to produce a better load factor. The load factor of 0.5 is quite reasonable for a on-demand air transportation company. The same load factor of 0.5 was also achieved by the manual schedule. However, while producing a manual schedule, the schedulers take into account the load factor. On the other hand, our model does not explicitly include the load factor, yet it has achieved the same load factor.

The model however significantly gains in terms of solution time, the total number of flight legs and the use appropriate aircraft type. The C208 is more comfortable and demanded by the tourists (which is what the model has predicted). Even though our model has achieved as solution in a short amount of time, the solution time can be decreased by using a programming language other than Matlab.

The technique that has been used to translate the problem into the mathematical formulation, in particular the use of heuristics, have certainly proved their usefulness. The model was solved in relatively quickly, which shows the impact of operations research in industrial problem solving.

A disadvantage of the heuristic used is that they may reduce the flexibility of the model and thereby affecting the optimality of the problem.

Appendix I: Airstrip names and their GPS co-ordinates

Camp name	***Abv.***	***GPS co-ordinates*** (*degrees minutes seconds*)
SLI	Selinda	(23°30' 45", -18°33'59")
XIG	Xigera	(22°44'18", -19°23'8")
MUB	Maun	(23°25'52", -19°58'21")
BBK	Kasane	(25°9'44", -17°49'58")
XOR	Lechwe	(23°5'38", -18°42'25")
CBE	Linyanti	(24°4'34", -18°31'17")
KWD	Lianshulu	(23°23'17", -18°12'32")
SWI	Moremi	(23°26'8", -19°11'50")
VUM	Vumburu	(22°48'53", -18°57'20")
JAO	Jacana	(22°35'42", -19°18'58")
CTB	Chitabe	(22°54'33", -19°18'40")
MOM	Little Mombo	(22°47'45", -19°12'31")
TSO	Jacks	(25°9'2", -20°27'23")
SRA	Seronga	(22°25'21", -18°49'23")
OMD	Duba Plains	(22°42'11", -19°1'3")
LVI	Livingstone	(25°49'12", -17°49'20")
SHN	Shinde	(23°9'47", -19°6'40")
HND	Tubu Tree	(22°28'31", -19°16'36")
ABU	Abu Camp	(22°32'55", -19°24'20")
VFA	Victoria Falls	(25°50'9", -18°5'55")
MLO	MLO	(25°1'44", -18°40'36")

Appendix II: Distance table

	SLI	**XIG**	**MUB**	**BBK**	**XOR**	**CBE**	**KWD**
SLI	0	125	157	201	49	63	42
XIG	125	0	101	320	85	177	149
MUB	157	101	0	306	146	177	196
BBK	201	320	306	0	250	143	202
XOR	49	85	146	250	0	111	64
CBE	63	177	177	143	111	0	84
KWD	42	149	196	202	64	84	0
SWI	71	80	86	245	66	104	110
VUM	89	49	132	289	42	148	105
JAO	132	18	118	330	88	187	151
CTB	107	21	94	300	70	157	134
MOM	107	21	110	304	65	162	129
TSO	278	293	199	292	300	246	318
SRA	125	72	170	324	76	187	127
OMD	103	41	134	304	56	162	118
LVI	270	384	357	73	319	209	274
SHN	72	56	100	264	46	121	103
HND	140	32	131	339	94	197	156
ABU	142	21	117	339	99	196	163
VFA	264	373	339	81	312	201	272
MLO	169	267	229	95	215	107	190

	CTB	MOM	TSO	SRA	OMD	LVI	SHN	HND	ABU	VFA	MLO	SWI	VUM	JAO
SLI	71	89	132	107	107	278	125	103	270	72	140	142	264	169
XIG	80	49	18	21	21	293	72	41	384	56	32	21	373	267
MUB	86	132	118	94	110	199	170	134	357	100	131	117	339	229
BBK	245	289	330	300	304	292	324	304	73	264	339	339	81	95
XOR	66	42	88	70	65	300	76	56	319	46	94	99	312	215
CBE	104	148	187	157	162	246	187	162	209	121	197	196	201	107
KWD	110	105	151	134	129	318	127	118	274	103	156	163	272	190
SWI	0	74	94	60	71	237	120	84	306	32	107	101	294	186
VUM	74	0	47	41	28	309	46	14	357	42	52	58	349	248
JAO	94	47	0	35	25	311	58	35	395	67	14	11	385	280
CTB	60	41	35	0	17	280	77	40	364	36	48	41	352	246
MOM	71	28	25	17	0	296	60	24	370	42	36	35	360	255
TSO	237	309	311	280	296	0	354	316	302	267	325	312	273	198
SRA	120	46	58	77	60	354	0	38	394	88	51	66	388	290
OMD	84	14	35	40	24	316	38	0	371	52	38	46	363	261
LVI	306	357	395	364	370	302	394	371	0	328	406	404	31	129
SHN	32	42	67	36	42	267	88	52	328	0	79	76	318	213
HND	107	52	14	48	36	325	51	38	406	79	0	16	396	292
ABU	101	58	11	41	35	312	66	46	404	76	16	0	393	287
VFA	294	349	385	352	360	273	388	363	31	318	396	393	0	110
MLO	186	248	280	246	255	198	290	261	129	213	292	287	110	0

Appendix III: ZIMPL model

set V := { read “costs.dat” as “<1n>” comment “#”};# of variables

set VG := { 1 to 4602 };# of ground variables

set R := { read “routes.dat” as “<1n>” comment “#”};# of routes

set RW := { 1 to 238 };# width of routes parameter

set A := { read “fleets.dat” as “<1n>” comment “#”};# of fleet

set ND := { 1 to 4602 };# width of nodes

set ND2 := { 1 to 4599 };# width of nodes

set TKW := { 1 to 4599 };# width of takeoffs parameter

set LDW := { 1 to 4599 };# width of landings parameter

set TGW := { 1 to 1 };# width of takeoffs ground parameter

set LGW := { 1 to 1 };# width of landings ground parameter

set GSINK1:= {1513 to 1533};# ground arcs to the super sink node

set GSINK2:= {3046 to 3066};# ground arcs to the super sink node

set GSINK3:= {4579 to 4599};# ground arcs to the super sink node

param c[V] := read “costs.dat” as “<1n> 2n” comment “#”; # cost for each variable

param rnum[R] := read “routes.dat” as “<1n> 2n” comment “#”; # of flights per route

param air[A] := read “fleets.dat” as “<1n> 2n” comment “#”; # of aircraft per fleet type

param tkn[ND] := read “takeoffs.dat” as “<1n> 2n” comment “#”; # of flights taking-off at a specific node

param lnd[ND] := read “landings.dat” as “<1n> 2n” comment “#”; # of flights landing at a specific node

param tgn[ND] := read “takeoffsgnd.dat” as “<1n> 2n” comment “#”; # of flights takeoff at a specific ground node

param lgn[ND] := read “landingsgnd.dat” as “<1n> 2n” comment “#”; # of flights landing at a specific ground node

include “routes.txt”; # Parameter table containing the variable number for each route.
include “takeoffs.txt”;
include “landings.txt”;
include “takeoffsgnd.txt”;
include “landingsgnd.txt”;

var x[V] binary;
var y[VG] integer;

minimize cost: sum <i> in V : c[i] * x[i];
subto routes: forall <i> in R do
sum <j> in { 1 to rnum[i]} : x[routes[i,j]] == 1;
subto flow: forall <i> in ND2 do
sum <j> in { 1 to tkn[i]} : x[takeoffs[i,j]]+ sum <k> in { 1 to tgn[i]} : y[takeoffsground[i,k]]
-
sum <w> in { 1 to lnd[i]} : x[landings[i,w]] - sum <q> in { 1 to lgn[i]} : y[landingsground[i,q]]
== 0;
subto flowend1: y[4600] - sum <g> in GSINK1: y[g] == 0;
subto flowend2: y[4601] - sum <g> in GSINK2: y[g] == 0;
subto flowend3: y[4602] - sum <g> in GSINK3: y[g] == 0;
subto flowstart1: y[4600] - sum <j> in {1 to tkn[4600]} : x[takeoffs[4600,j]] == 0;
subto flowstart2: y[4601] - sum <j> in {1 to tkn[4601]} : x[takeoffs[4601,j]] == 0;
subto flowstart3: y[4602] - sum <j> in {1 to tkn[4602]} : x[takeoffs[4602,j]] == 0;
subto avail1: y[4600] <= air[1];

subto avail2: y[4601] <= air[2];

subto avail3: y[4602] <= air[3];

subto aircraft: x[2381] + x[2382] + x[2383] + x[2384] + x[2385] + x[2386] + x[2387] + x[2388] + x[2389] + x[2390] + x[2391] + x[2392] + x[2393] <= 13;

End of File

Appendix IV: The flight schedule for 25 flight bookings

Aircraft	***Etd***	***From***	***Eta***	***To***	***Pax***	***BI***
A2-ADK	12:50	MUB	13:50	TSO	2	14
Aircraft	***Etd***	***From***	***Eta***	***To***	***Pax***	***BI***
A2-AIV	12:50	MUB	13:25	CTB	3	15, 21
	13:25	CTB	13:50	VUM	3	8, 21
	14:10	VUM	14:30	XOR	0	EMPTY
	14:30	XOR	15:00	XIG	1	1
Aircraft	***Etd***	***From***	***Eta***	***To***	***Pax***	***BI***
A2-EGL	7:00	CBE	7:50	MUB	2	5, 24
Aircraft	***Etd***	***From***	***Eta***	***To***	***Pax***	***BI***
A2-FOX	7:40	MUB	7:40	MOM	1	22
	11:10	MOM	11:40	SHN	0	EMPTY
	11:40	SHN	12:10	MOM	2	20
	12:10	MOM	13:00	CBE	2	11
	13:00	CBE	13:50	BBK	2	9
	13:50	BBK	14:40	CBE	3	2, 3
	14:40	CBE	15:30	VUM	2	6
	15:40	VUM	16:20	SRA	0	EMPTY
	16:20	SRA	16:50	OMD	1	17
	16:50	OMD	17:20	SRA	2	18, 19
	17:20	SRA	18:00	VUM	1	16
Aircraft	***Etd***	***From***	***Eta***	***To***	***Pax***	***BI***
A2-JKL	6:00	BBK	7:00	SLI	1	23
	7:00	SLI	7:20	KWD	1	26
	14:30	KWD	15:10	SWI	5	4
	15:10	SWI	15:40	MUB	1	25
Aircraft	***Etd***	***From***	***Eta***	***To***	***Pax***	***BI***
A2-OWL	12:50	VUM	13:40	MUB	2	10
	13:50	MUB	14:30	JAO	4	7
	14:30	JAO	15:30	CBE	2	13
	15:30	CBE	16:40	TSO	2	12

Appendix V: The flight schedule for 50 flight bookings

Aircraft	***Etd***	***From***	***Eta***	***To***	***Pax***	***BI***
A2-ADK	12:50	MUB	13:30	JAO	4	7
	13:30	JAO	14:30	CBE	2	13
	14:30	CBE	15:20	BBK	2	9
	15:20	BBK	16:10	CBE	4	39
Aircraft	***Etd***	***From***	***Eta***	***To***	***Pax***	***BI***
A2-AIV	12:50	MUB	13:25	CTB	4	15, 21, 26
	13:25	CTB	13:50	VUM	4	8, 21, 26
	13:50	VUM	14:10	XOR	0	EMPTY
	14:10	XOR	14:40	XIG	2	1, 30
	14:40	XIG	15:00	CTB	0	EMPTY
	15:20	CTB	15:40	JAO	2	33
	15:40	JAO	16:20	MUB	2	27, 28
Aircraft	***Etd***	***From***	***Eta***	***To***	***Pax***	***BI***
A2-ANT	11:10	HND	11:40	VUM	4	37
	13:20	VUM	14:10	MUB	2	10
	14:10	MUB	14:50	XIG	4	40
	15:00	XIG	15:20	ABU	0	EMPTY
	15:30	ABU	16:10	MUB	6	43
Aircraft	***Etd***	***From***	***Eta***	***To***	***Pax***	***BI***
A2-BEE	13:20	MUB	14:20	TSO	2	14
Aircraft	***Etd***	***From***	***Eta***	***To***	***Pax***	***BI***
A2-EGL	7:00	CBE	7:50	MUB	2	5, 24
	7:50	MUB	8:20	SWI	1	29
	8:20	SWI	9:30	BBK	12	32
	10:00	BBK	11:00	KWD	12	31
	11:00	KWD	11:40	SWI	5	4
	11:40	SWI	12:10	MUB	1	25

Aircraft	***Etd***	***From***	***Eta***	***To***	***Pax***	***BI***
A2-FOX	7:20	MUB	8:00	MOM	1	22
	13:30	MOM	14:10	MUB	4	34, 36
	14:10	MUB	15:00	OMD	2	44
	15:00	OMD	15:30	SRA	0	EMPTY
	15:30	SRA	16:00	OMD	1	17
	16:00	OMD	16:30	SRA	2	18, 19
	16:40	SRA	17:20	VUM	6	16, 46, 47, 48, 49, 50
Aircraft	***Etd***	***From***	***Eta***	***To***	***Pax***	***BI***
A2-JKL	6:00	BBK	7:00	SLI	1	23
	11:00	SLI	11:40	XIG	2	41
	11:40	XIG	13:10	BBK	2	42
	13:10	BBK	13:40	LVI	2	45
	13:40	LVI	14:10	BBK	2	38
	14:10	BBK	15:00	CBE	3	2, 3
Aircraft	***Etd***	***From***	***Eta***	***To***	***Pax***	***BI***
A2-OWL	6:10	VUM	6:30	SHN	0	EMPTY
	11:00	SHN	11:20	MOM	2	20
	11:20	MOM	12:10	CBE	2	11
	12:10	CBE	12:55	VUM	4	6, 12
	12:55	VUM	14:20	TSO	4	12, 35

Appendix VI: The flight schedule for 75 flight bookings

Aircraft	***Etd***	***From***	***Eta***	***To***	***Pax***	***BI***
A2-ADK	6:00	MUB	6:30	SWI	0	EMPTY
	8:20	SWI	9:30	BBK	5	60
	13:40	BBK	14:25	CBE	4	2, 3, 23
	14:25	CBE	14:50	SLI	4	23, 70
	14:50	SLI	15:30	XIG	2	41
Aircraft	***Etd***	***From***	***Eta***	***To***	***Pax***	***BI***
A2-AIV	13:50	MUB	14:30	XIG	2	40
	14:40	XIG	15:00	ABU	0	EMPTY
	15:00	ABU	15:40	MUB	6	43
Aircraft	***Etd***	***From***	***Eta***	***To***	***Pax***	***BI***
A2-ANT	7:50	HND	8:10	JAO	0	EMPTY
	8:10	JAO	8:50	MUB	5	27, 28, 69
	13:50	MUB	14:30	MOM	3	22, 71
	14:30	MOM	15:20	CBE	2	11
	15:20	CBE	16:10	BBK	2	9
Aircraft	***Etd***	***From***	***Eta***	***To***	***Pax***	***BI***
A2-BEE	13:50	MUB	14:40	OMD	2	44
Aircraft	***Etd***	***From***	***Eta***	***To***	***Pax***	***BI***
A2-BUF	7:50	MUB	8:20	SWI	1	29
	8:20	SWI	9:30	BBK	12	32
	10:50	BBK	11:50	KWD	12	31
	11:50	KWD	12:30	SWI	5	4
	12:30	SWI	13:00	MUB	1	25

Aircraft	***Etd***	***From***	***Eta***	***To***	***Pax***	***BI***
A2-EGL	7:10	CBE	8:00	MUB	2	5, 24
	11:10	MUB	11:50	SHN	0	EMPTY
	11:50	SHN	12:10	MOM	2	20
	12:10	MOM	12:30	JAO	0	EMPTY
	12:30	JAO	12:50	CTB	4	64
	12:50	CTB	13:10	VUM	4	8, 61
	13:10	VUM	13:25	MOM	7	10, 56, 61, 66
	13:25	MOM	14:00	MUB	7	10, 34, 36, 61, 66
	14:30	MUB	15:10	HND	5	21, 26, 58, 72
	15:10	HND	15:30	VUM	11	21, 26, 37, 58, 68

Aircraft	***Etd***	***From***	***Eta***	***To***	***Pax***	***BI***
A2-FOX	7:50	MUB	8:30	XIG	0	EMPTY
	8:30	XIG	9:10	MUB	2	62
	12:50	MUB	13:30	CTB	8	7, 15, 73
	13:30	CTB	14:00	JAO	6	7, 33
	14:00	JAO	14:50	CBE	2	13
	14:50	CBE	15:40	OMD	2	67
	15:40	OMD	16:10	SRA	4	18, 19, 53, 54
	16:10	SRA	16:50	VUM	8	16, 46, 47, 48, 49, 50, 55, 57
	16:50	VUM	17:30	SRA	2	51, 75
	17:30	SRA	18:00	OMD	2	17, 52

Aircraft	***Etd***	***From***	***Eta***	***To***	***Pax***	***BI***
A2-OWL	7:50	VUM	8:10	XOR	0	EMPTY
	11:00	XOR	11:30	XIG	3	1, 30, 59
	11:30	XIG	13:00	BBK	2	42
	14:00	BBK	14:30	LVI	2	45

Aircraft	***Etd***	***From***	***Eta***	***To***	***Pax***	***BI***
A2-ZEB	10:00	LVI	10:30	BBK	2	38
	11:00	BBK	11:50	CBE	4	39
	11:50	CBE	12:40	VUM	2	6
	12:40	VUM	13:25	CBE	4	35, 63
	13:25	CBE	14:15	MUB	6	12, 35, 65
	14:15	MUB	15:10	TSO	6	12, 14, 35

Appendix VII: The flight schedule for 100 flight bookings

Aircraft	***Etd***	***From***	***Eta***	***To***	***Pax***	***BI***
A2-ADK	6:30	MUB	6:30	SWI	3	29, 97, 98
	7:00	SWI	8:10	BBK	5	60
	13:30	BBK	14:00	LVI	2	45
	14:20	LVI	14:50	BBK	2	38, 82
Aircraft	***Etd***	***From***	***Eta***	***To***	***Pax***	***BI***
A2-AIV	6:00	MUB	6:40	XIG	0	EMPTY
	8:10	XIG	8:50	MUB	2	62
	13:40	MUB	14:40	TSO	2	14
Aircraft	***Etd***	***From***	***Eta***	***To***	***Pax***	***BI***
A2-ANT	6:30	HND	6:50	JAO	0	EMPTY
	11:00	JAO	12:00	CBE	2	13
	12:00	CBE	14:40	TSO	2	6, 12, 35
Aircraft	***Etd***	***From***	***Eta***	***To***	***Pax***	***BI***
A2-BUF	6:40	MUB	7:10	SWI	0	EMPTY
	7:10	SWI	8:20	BBK	12	32
	10:00	BBK	11:00	KWD	12	31
	11:00	KWD	11:40	SWI	5	4
	11:40	SWI	12:00	SHN	0	EMPTY
	12:00	SHN	12:20	MOM	2	20
	12:20	MOM	12:40	CTB	0	EMPTY
	12:50	CTB	13:10	VUM	4	8, 61
	13:10	VUM	13:25	MOM	8	10, 56, 61, 66, 93
	13:25	MOM	14:00	MUB	10	10, 34, 36, 56, 61, 93
	14:20	MUB	15:00	XIG	5	40, 96
	15:00	XIG	15:20	ABU	0	EMPTY
	15:30	ABU	16:10	MUB	6	43

Aircraft	*Etd*	*From*	*Eta*	*To*	*Pax*	*BI*
A2-EGL	7:00	CBE	7:50	MUB	2	
	12:50	SWI	13:20	CTB	8	7, 15
	13:20	CTB	13:40	JAO	6	7, 33, 73
	13:50	JAO	14:10	CTB	4	64
	14:20	CTB	14:50	SWI	0	EMPTY
	15:50	SWI	16:20	MUB	1	25

Aircraft	*Etd*	*From*	*Eta*	*To*	*Pax*	*BI*
A2-GNU	13:10	MUB	13:45	MOM	8	22, 44, 71, 83, 84, 94
	13:45	MOM	14:30	CBE	5	11, 44, 94
	14:30	CBE	15:20	OMD	4	44, 67
	15:20	OMD	15:40	SRA	4	18, 19, 53, 54
	17:00	SRA	17:20	VUM	11	16, 46, 47, 48, 49, 50, 55, 57, 76, 78, 79

Aircraft	*Etd*	*From*	*Eta*	*To*	*Pax*	*BI*
A2-FOX	6:00	MUB	6:50	VUM	6	21, 26, 58, 80, 81, 86
	6:50	VUM	7:30	SRA	6	51, 75, 88, 89, 90, 100
	7:30	SRA	8:00	OMD	3	17, 52, 77
	8:00	OMD	8:30	JAO	0	EMPTY
	8:30	JAO	9:20	MUB	7	27, 28, 69, 91, 92
	12:30	MUB	13:20	HND	2	72
	13:20	HND	14:00	VUM	8	37, 68
	14:00	VUM	14:50	CBE	2	63
	14:50	CBE	15:35	BBK	2	9
	15:35	BBK	16:20	CBE	5	39, 95
	16:20	CBE	17:10	MUB	7	65, 99

Aircraft	*Etd*	*From*	*Eta*	*To*	*Pax*	*BI*
A2-OWL	6:00	VUM	6:20	XOR	0	EMPTY
	11:00	XOR	11:30	XIG	3	1, 30, 59
	11:40	XIG	13:10	BBK	2	42
	14:10	BBK	14:55	CBE	4	2, 3, 23
	14:55	CBE	15:20	SLI	3	23, 70
	15:30	SLI	16:10	XIG	2	41

Aircraft	*Etd*	*From*	*Eta*	*To*	*Pax*	*BI*
A2-ZEB	12:40	LVI	13:00	VFA	0	EMPTY
	13:00	VFA	13:40	MLO	12	85

Appendix VIII: The flight schedule for 125 flight bookings

Aircraft	***Etd***	***From***	***Eta***	***To***	***Pax***	***BI***
A2-ADK	12:50	MUB	13:30	MOM	5	22, 71, 83, 84
	13:30	MOM	14:00	SRA	0	EMPTY
	14:10	SRA	14:30	OMD	1	103
	14:30	OMD	14:50	SRA	5	18, 19, 53, 54, 125
	15:20	SRA	15:40	OMD	4	17, 52, 77, 124
Aircraft	***Etd***	***From***	***Eta***	***To***	***Pax***	***BI***
A2-ANT	6:00	HND	6:20	JAO	0	EMPTY
	11:00	JAO	12:00	CBE	2	13
	12:20	CBE	13:05	VUM	5	6, 121
	13:05	VUM	13:50	CBE	6	35, 63
	13:50	CBE	15:00	TSO	4	12, 35
Aircraft	***Etd***	***From***	***Eta***	***To***	***Pax***	***BI***
A2-BUF	6:00	MUB	6:30	SWI	3	29, 97, 98
	7:00	SWI	8:10	BBK	5	60
	10:00	BBK	11:00	KWD	2	106
	11:00	KWD	11:40	SWI	12	107
	11:40	SWI	12:00	SHN	0	EMPTY
	12:00	SHN	12:20	MOM	2	20
	12:20	MOM	13:05	CBE	4	11, 120
	13:05	CBE	13:50	BBK	6	9, 109, 120
	13:50	BBK	14:20	LVI	4	45, 110
	14:20	LVI	14:50	BBK	8	32, 82, 114

Aircraft	*Etd*	*From*	*Eta*	*To*	*Pax*	*BI*
A2-EGL	7:00	CBE	7:50	MUB	3	108
	10:50	MUB	11:30	SLI	0	EMPTY
	11:30	SLI	12:10	XIG	3	41
	12:20	XIG	12:30	JAO	0	EMPTY
	12:30	JAO	12:50	CTB	4	64
	12:50	CTB	13:10	VUM	4	8, 61
	13:10	VUM	13:30	MOM	8	10, 56, 61, 66, 93
	13:30	MOM	14:00	MUB	10	10, 34, 36, 56, 93, 116
	14:20	MUB	15:00	VUM	9	21, 26, 58, 80, 81, 86, 104, 119
	15:50	VUM	16:10	SRA	10	51, 75, 88, 89, 90, 100, 102, 122, 123
	17:10	SRA	17:30	VUM	11	16, 46, 47, 48, 49, 50, 55, 57, 76, 78, 79

Aircraft	*Etd*	*From*	*Eta*	*To*	*Pax*	*BI*
A2-GNU	6:00	MUB	6:30	SWI	0	EMPTY
	7:00	SWI	8:10	BBK	12	32
	13:30	BBK	14:15	CBE	6	2, 3, 23, 113
	14:15	CBE	15:00	OMD	7	23, 67, 70, 113
	15:00	OMD	15:30	SLI	7	23, 70, 113, 115
	15:30	SLI	16:30	BBK	4	112

Aircraft	*Etd*	*From*	*Eta*	*To*	*Pax*	*BI*
A2-FOX	6:00	MUB	6:50	CBE	0	EMPTY
	7:00	CBE	7:50	MUB	2	5, 24
	7:50	MUB	8:30	XIG	0	EMPTY
	8:30	XIG	9:10	MUB	2	62
	9:30	MUB	10:10	XIG	0	EMPTY
	10:10	XIG	11:20	BBK	2	42
	11:20	BBK	12:20	KWD	2	106
	12:50	KWD	13:30	SWI	5	4
	13:30	SWI	14:10	MUB	1	25
	14:10	MUB	14:50	XIG	5	40, 96
	14:50	XIG	15:20	ABU	0	EMPTY
	15:20	ABU	16:00	MUB	6	43
	16:10	MUB	17:00	CBE	3	94, 118

Aircraft	***Etd***	***From***	***Eta***	***To***	***Pax***	***BI***
A2-JKL	13:00	BBK	13:45	CBE	5	39, 95
	13:45	CBE	14:40	MUB	4	65, 95, 99
Aircraft	***Etd***	***From***	***Eta***	***To***	***Pax***	***BI***
A2-LEO	6:00	MUB	6:40	JAO	0	EMPTY
	8:00	JAO	8:40	MUB	5	27, 28, 91, 92, 69
	12:30	MUB	13:10	HND	2	72
	13:10	HND	13:40	VUM	8	37, 68
	13:40	VUM	14:00	XOR	0	EMPTY
	14:00	XOR	14:30	XIG	3	1, 30, 59
	14:30	XIG	14:50	ABU	0	EMPTY
	14:50	ABU	15:30	MUB	7	117
	15:30	MUB	16:00	CTB	10	7, 15, 44, 73
	16:00	CTB	16:20	JAO	8	7, 33, 44
	16:20	JAO	16:40	OMD	3	44, 105
Aircraft	***Etd***	***From***	***Eta***	***To***	***Pax***	***BI***
A2-OWL	8:00	VUM	8:50	MUB	3	111
	13:10	MUB	14:10	TSO	2	14
Aircraft	***Etd***	***From***	***Eta***	***To***	***Pax***	***BI***
	12:20	LVI	12:40	VFA	0	EMPTY
	12:40	VFA	13:20	MLO	12	85

Appendix IX: The flight schedule for 150 flight bookings

Aircraft	***Etd***	***From***	***Eta***	***To***	***Pax***	***BI***
A2-ADK	6:00	MUB	6:40	XIG	0	EMPTY
	8:10	XIG	8:50	MUB	2	62
	13:30	MUB	14:30	TSO	1	137
Aircraft	***Etd***	***From***	***Eta***	***To***	***Pax***	***BI***
A2-AIV	6:00	MUB	7:00	CBE	3	94, 141, 143
	7:00	CBE	8:00	MUB	3	108
	13:40	MUB	14:40	TSO	2	14
Aircraft	***Etd***	***From***	***Eta***	***To***	***Pax***	***BI***
A2-ANT	6:00	HND	6:20	JAO	0	EMPTY
	11:30	JAO	12:25	CBE	3	13, 105
	12:25	CBE	13:15	MUB	6	65, 67, 99, 105
	13:15	MUB	14:00	OMD	5	44, 67, 105
	14:00	OMD	14:20	SRA	0	EMPTY
	14:30	SRA	14:50	OMD	1	103
	14:50	OMD	15:30	SLI	2	115
	15:30	SLI	16:30	BBK	4	112
Aircraft	***Etd***	***From***	***Eta***	***To***	***Pax***	***BI***
A2-BEE	13:50	MUB	14:30	XIG	5	40, 96
	14:30	XIG	15:10	JAO	0	EMPTY
	15:10	JAO	15:30	CTB	4	64
	15:30	CTB	15:50	VUM	2	8
Aircraft	***Etd***	***From***	***Eta***	***To***	***Pax***	***BI***
A2-GNU	7:30	MUB	8:10	JAO	0	EMPTY
	8:10	JAO	8:50	MUB	10	27, 28, 69, 91, 92, 147, 148, 150, 169
	12:50	MUB	13:30	VUM	7	119, 132, 133, 135, 136, 145
	13:30	VUM	13:50	MOM	6	10, 56, 66, 93
	13:50	MOM	14:25	MUB	12	10, 34, 36, 56, 93, 116
	14:25	MUB	15:00	MOM	8	22, 71, 83, 84, 149

Aircraft	*Etd*	*From*	*Eta*	*To*	*Pax*	*BI*
A2-BUF	13:50	MUB	6:30	SWI	0	EMPTY
	7:00	SWI	8:10	BBK	12	32
	10:00	BBK	11:00	KWD	12	31
	11:00	KWD	11:40	SWI	5	4
	11:40	SWI	12:00	SHN	0	EMPTY
	12:00	SHN	12:20	MOM	2	20
	12:20	MOM	13:05	CBE	4	11, 120
	13:05	CBE	13:50	BBK	6	9, 109, 120
	13:50	BBK	14:50	KWD	2	106
	15:00	KWD	15:40	SWI	12	107
	16:30	SWI	17:00	MUB	1	25

Aircraft	*Etd*	*From*	*Eta*	*To*	*Pax*	*BI*
A2-EGL	7:40	CBE	8:30	MUB	2	5, 24
	8:40	MUB	9:20	VUM	12	21, 26, 58, 80, 81, 86, 104 127,128, 129, 130, 131
	9:20	VUM	9:40	XOR	0	EMPTY
	11:30	XOR	12:00	XIG	3	1, 30, 59
	12:00	XIG	13:20	BBK	2	42
	13:20	BBK	13:50	LVI	4	45, 110
	13:50	LVI	14:20	BBK	8	38, 82, 114
	14:20	BBK	15:05	CBE	6	2, 3, 23, 113
	15:05	CBE	15:30	SLI	5	23, 70, 113
	15:30	SLI	16:10	XIG	2	41

Aircraft	*Etd*	*From*	*Eta*	*To*	*Pax*	*BI*
A2-FOX	6:30	MUB	7:10	SWI	4	29, 97, 98, 139
	7:10	SWI	8:10	BBK	6	60, 139
	11:00	BBK	12:10	ABU	4	95, 138, 142
	12:20	ABU	13:00	MUB	8	43, 95, 142
	13:00	MUB	13:40	CTB	4	15, 73
	13:40	CTB	14:20	MUB	7	33, 61, 140, 144, 146
	14:20	MUB	15:00	JAO	6	7, 33
	15:00	JAO	15:30	ABU	0	EMPTY
	15:30	ABU	16:10	MUB	7	117
	16:30	MUB	17:20	CBE	2	118

Aircraft	*Etd*	*From*	*Eta*	*To*	*Pax*	*BI*
A2-JKL	11:00	BBK	11:50	CBE	4	39
	11:50	CBE	12:40	VUM	1	6, 121
	12:40	VUM	13:25	CBE	4	35, 63
	13:25	CBE	14:40	TSO	4	12, 35

Aircraft	***Etd***	***From***	***Eta***	***To***	***Pax***	***BI***
A2-LEO	12:50	MUB	13:30	HND	2	72
	14:40	HND	15:10	VUM	8	37, 68
	15:10	VUM	15:30	SRA	10	51, 75, 88, 89, 90, 100, 101, 102, 122, 123
	15:40	SRA	16:00	OMD	4	17, 52, 77, 124
	17:00	OMD	17:20	SRA	6	18, 19, 53, 54, 125, 126
	17:40	SRA	18:00	VUM	11	16, 46, 47, 48, 49, 50, 55, 57, 76, 78, 79
Aircraft	***Etd***	***From***	***Eta***	***To***	***Pax***	***BI***
A2-OWL	7:50	VUM	8:40	MUB	3	111
Aircraft	***Etd***	***From***	***Eta***	***To***	***Pax***	***BI***
A2-ZEB	6:40	LVI	7:00	VFA	0	EMPTY
	12:20	VFA	13:00	MLO	10	85

Appendix X: The flight schedule for 175 flight bookings

Aircraft	***Etd***	***From***	***Eta***	***To***	***Pax***	***BI***
A2-ADK	12:50	MUB	13:30	CTB	4	15, 73
	13:30	CTB	13:50	JAO	5	33, 158
	13:50	JAO	14:10	HND	0	EMPTY
	14:10	HND	15:00	MUB	2	155, 165
Aircraft	***Etd***	***From***	***Eta***	***To***	***Pax***	***BI***
A2-AIV	13:10	MUB	14:10	TSO	1	137
Aircraft	***Etd***	***From***	***Eta***	***To***	***Pax***	***BI***
A2-ANT	8:10	HND	8:30	XIG	0	EMPTY
	10:10	XIG	11:40	BBK	2	42
	11:40	BBK	12:40	KWD	2	106
	12:40	KWD	13:20	SWI	5	4
	13:20	SWI	13:50	MUB	3	25, 166, 167
Aircraft	***Etd***	***From***	***Eta***	***To***	***Pax***	***BI***
A2-BEE	13:10	MUB	14:10	TSO	2	14
Aircraft	***Etd***	***From***	***Eta***	***To***	***Pax***	***BI***
A2-ADK	12:50	MUB	13:30	CTB	4	15, 73
	13:30	CTB	13:50	JAO	5	33, 158
	13:50	JAO	14:10	HND	0	EMPTY
	14:10	HND	15:00	MUB	2	155, 165
Aircraft	***Etd***	***From***	***Eta***	***To***	***Pax***	***BI***
A2-AIV	13:10	MUB	14:10	TSO	1	137
Aircraft	***Etd***	***From***	***Eta***	***To***	***Pax***	***BI***
A2-ANT	8:10	HND	8:30	XIG	0	EMPTY
	10:10	XIG	11:40	BBK	2	42
	11:40	BBK	12:40	KWD	2	106
	12:40	KWD	13:20	SWI	5	4
	13:20	SWI	13:50	MUB	3	25, 166, 167

Aircraft	*Etd*	*From*	*Eta*	*To*	*Pax*	*BI*
A2-BEE	13:10	MUB	14:10	TSO	2	14

Aircraft	*Etd*	*From*	*Eta*	*To*	*Pax*	*BI*
A2-BUF	6:00	MUB	6:40	JAO	0	EMPTY
	8:00	JAO	8:40	MUB	12	27, 28, 69, 91, 92, 147, 148, 150, 154, 156
	10:40	MUB	10:40	VUM	12	21, 26, 58, 80, 81, 86, 104, 127, 128, 129, 130, 131
	11:20	VUM	12:10	CBE	2	63
	13:10	CBE	14:00	MUB	5	65, 67, 99
	14:00	MUB	14:35	JAO	8	7, 44, 67
	14:35	JAO	15:00	OMD	6	44, 67, 105, 175
	15:00	OMD	15:20	SRA	0	EMPTY
	15:20	SRA	15:40	OMD	4	17, 77, 52, 124
	15:40	OMD	16:00	SRA	6	18, 19, 53, 54, 125, 126
	16:00	SRA	16:20	VUM	11	16, 46, 47, 48, 49, 50, 55, 57, 76, 78, 79

Aircraft	*Etd*	*From*	*Eta*	*To*	*Pax*	*BI*
A2-EGL	7:10	CBE	8:00	MUB	4	5, 24, 162, 168
	8:00	MUB	8:40	SHN	0	EMPTY
	11:00	SHN	11:20	MOM	2	20
	11:20	MOM	11:35	JAO	5	11, 120, 174
	11:35	JAO	12:35	CBE	6	11, 13, 120
	12:35	CBE	13:10	BBK	6	9, 109, 120
	13:10	BBK	13:40	LVI	4	45, 110
	14:10	LVI	14:10	BBK	8	38, 82, 114
	14:10	BBK	14:55	CBE	6	2, 3, 23, 113
	14:55	CBE	15:20	SLI	5	23, 70, 113
	15:20	SLI	16:20	BBK	4	112

Aircraft	*Etd*	*From*	*Eta*	*To*	*Pax*	*BI*
A2-GNU	6:20	MUB	6:50	SWI	0	EMPTY
	8:10	SWI	9:20	BBK	12	32
	10:00	SWI	11:00	KWD	12	31
	11:00	KWD	11:40	SWI	12	107
	11:40	SWI	12:10	XOR	0	EMPTY
	12:10	XOR	12:40	XIG	3	1, 30, 59
	12:40	XIG	13:00	JAO	0	EMPTY
	13:00	JAO	13:20	CTB	4	64
	13:20	CTB	13:40	VUM	7	8, 61, 140, 144, 146
	13:40	VUM	14:20	MUB	10	10, 56, 61, 93, 140, 144, 146, 152
	14:20	MUB	15:00	VUM	9	119, 132, 135, 136, 145, 159, 163, 170,
	15:00	VUM	15:20	MOM	3	66, 160

Aircraft	*Etd*	*From*	*Eta*	*To*	*Pax*	*BI*
A2-FOX	6:10	MUB	7:00	CBE	4	94, 141, 143, 157
	7:00	CBE	7:50	MUB	3	108
	7:50	MUB	8:30	XIG	0	EMPTY
	8:30	XIG	9:10	MUB	2	62
	12:50	MUB	13:30	MOM	8	22, 71, 83, 84, 149, 153, 171
	13:30	MOM	14:10	MUB	6	34, 36, 116
	14:20	MUB	15:00	XIG	6	40, 96, 169
	15:00	XIG	15:30	ABU	0	EMPTY
	15:30	ABU	16:10	MUB	7	117
	16:10	MUB	17:00	CBE	2	118

Aircraft	*Etd*	*From*	*Eta*	*To*	*Pax*	*BI*
A2-JKL	11:00	BBK	11:50	CBE	4	39
	11:50	CBE	12:35	VUM	5	6, 12, 121
	11:50	VUM	14:00	TSO	4	12, 35

Aircraft	*Etd*	*From*	*Eta*	*To*	*Pax*	*BI*
A2-LEO	6:30	MUB	7:00	SWI	4	29, 97, 98, 139
	7:00	SWI	8:10	BBK	6	60, 139

Aircraft	*Etd*	*From*	*Eta*	*To*	*Pax*	*BI*
A2-LEO	11:00	BBK	12:25	ABU	4	95, 138, 142
	12:25	ABU	13:00	MUB	8	43, 95, 142
	13:00	MUB	13:40	HND	2	72
	13:40	HND	13:40	VUM	8	37, 68
	14:10	VUM	14:30	SRA	12	51, 75, 88, 89, 90, 100, 101, 102, 122, 123, 151, 172
	14:30	SRA	14:50	OMD	2	103, 173
	14:50	OMD	15:30	SLI	2	115
	15:30	SLI	16:10	XIG	2	41

Aircraft	*Etd*	*From*	*Eta*	*To*	*Pax*	*BI*
A2-OWL	7:50	VUM	8:40	MUB	3	111

Aircraft	*Etd*	*From*	*Eta*	*To*	*Pax*	*BI*
AZEB	6:00	LVI	6:20	VFA	0	EMPTY
	12:20	VFA	13:00	MLO	12	85
	14:20	MLO	15:00	VFA	12	115

Appendix XI: Manual schedule

Aircraft	***Etd***	***From***	***Eta***	***To***	***Pax***	***Group***
A2-ADK	14:30	MUB	15:05	JAO	4	x Merrill 7nt
A2-ADK	15:15	JAO	15:50	MUB	1	x NAS-Staff-Isaac
A2-ADK					1	x nas-staff-keowetse
A2-ADK					1	x NAS-Staff-Mike
A2-ADK					1	x NAS-Staff-Nelly
A2-ADK					1	x Nas-Staff-Sam

Aircraft	***Etd***	***From***	***Eta***	***To***	***Pax***	***Group***
A2-AIV	7:50	MUB	8:20	XIG	1	x Staff-Sefofane-Warren
A2-AIV	8:30	XIG	9:00	MUB	2	x Marrero
A2-AIV					1	x Staff-Sefofane-Warren
A2-AIV	10:45	MUB	11:30	VUM	1	x Staff-OWS-Martha
A2-AIV	11:40	VUM	12:15	CBE	2	x Trimble 7nt
A2-AIV	12:25	CBE	13:45	TSO	2	x Luecke
A2-AIV					2	x Trimble 7nt
A2-AIV	13:55	TSO	14:50	MUB	0	Empty

Aircraft	***Etd***	***From***	***Eta***	***To***	***Pax***	***Group***
A2-ANT	8:10	HND	8:15	JAO	1	x NAS-Staff-Lebo
A2-ANT	8:25	JAO	9:00	MUB	3	x Auster 7nt
A2-ANT					1	1 x NAS-Staff-Lebo
A2-ANT	13:00	MUB	13:55	TSO	2	x Farley
A2-ANT					1	x Farley Extra Seat Luggage

Aircraft	***Etd***	***From***	***Eta***	***To***	***Pax***	***Group***
A2-EGL	7:40	CBE	8:15	MUB	3	x Boehmke/1
A2-EGL					1	x Boehmke/2
A2-EGL					1	x Staff-Sable-Lee
A2-EGL					1	x Staff-Sable-Raymond
A2-EGL					1	x Staff-Sefofane-Modiri
A2-EGL	9:30	MUB	9:55	ABU	1	x Staff-Sefofane-Faadhil

Aircraft	*Etd*	*From*	*Eta*	*To*	*Pax*	*Group*
A2-BEE	7:30	MUB	7:45	SWI	0	Empty
A2-BEE	8:00	SWI	9:15	BBK	5	x BOX#69(08)/2
A2-BEE	10:00	BBK	11:05	KWD	2	x BOX#73(08)/2
A2-BEE	11:20	KWD	12:05	SWI	5	x BOX#71(08)/2
A2-BEE	12:15	SWI	12:25	CTB	0	Empty
A2-BEE	12:35	CTB	13:00	JAO	3	x Kirby 7nt
A2-BEE	13:10	JAO	13:35	CTB	4	x Higbee Stacy
A2-BEE	13:45	CTB	14:05	MUB	2	x Spilsbury
A2-BEE					1	xStaff-Flamingo-Lebopo
A2-BEE					1	x Staff-Flamingo-Simon
A2-BEE	14:35	MUB	15:05	XIG	4	x Wright
A2-BEE	15:15	XIG	15:25	HND	0	Empty

Aircraft	*Etd*	*From*	*Eta*	*To*	*Pax*	*Group*
A2-BOK	9:00	MUB	9:30	XIG	1	x Staff-GREAT Explo-Goweditswe
A2-BOK					1	x Staff-Great Explo-OT
A2-BOK	9:40	XIG	11:15	BBK	2	x Harris
A2-BOK	11:55	BBK	12:55	SLI	0	Empty
A2-BOK	13:05	SLI	14:05	BBK	4	x Stahl
A2-BOK	14:10	BBK	15:05	CBE	1	x Z3-10(08)Guide Richard
A2-BOK					2	x Z3-10(08)Kauth
A2-BOK	15:15	CBE	16:05	MUB	1	x Staff-Sefofane-Lekani

Aircraft	*Etd*	*From*	*Eta*	*To*	*Pax*	*Group*
A2-BUF	7:45	MUB	8:00	SWI	1	x Staff-OWS-KG
A2-BUF					1	Staff-OWS-KK
A2-BUF					1	x Staff-Ows-Tirelo
A2-BUF	8:20	SWI	9:15	BBK	12	x BOX#69(08)/1
A2-BUF	10:00	BBK	10:50	KWD	12	x BOX#73(08)/1
A2-BUF	11:10	KWD	11:45	SWI	12	x BOX#71(08)/1
A2-BUF	11:55	SWI	12:10	MUB	1	x Staff-OWS-KP
A2-BUF					1	x Staff-OWS-Lindi
A2-BUF					1	x Staff-OWS-Sam
A2-BUF	13:10	MUB	13:25	CTB	2	x Freeman Kalm
A2-BUF					2	x Mahul Shah
A2-BUF					2	x Sollek & McNally
A2-BUF					1	x Staff-Sefofane-Witness
A2-BUF	13:35	CTB	13:55	JAO	2	x Holden
A2-BUF					2	x Mahul Shah
A2-BUF					1	x Staff-Sefofane-Witness
A2-BUF					2	x Thomas & Lorenz
A2-BUF	14:05	JAO	14:15	VUM	2	x Holden
A2-BUF					2	x Mahul Shah
A2-BUF					1	x NAS-Staff-Joanne
A2-BUF					1	x Staff-Nas-Clara
A2-BUF					1	x STAFF-NAS-CLINT
A2-BUF					1	x Staff-Nas-Dona
A2-BUF					1	x Staff-Nas-Graeme
A2-BUF					1	x Staff-OWS-Russel
A2-BUF					1	x Staff-Sefofane-Witness

Aircraft	*Etd*	*From*	*Eta*	*To*	*Pax*	*Group*
A2-BUF	14:25	VUM	14:30	OMD	2	x Mahul Shah
A2-BUF					1	x NAS-Staff-Joanne
A2-BUF					1	x Staff-Nas-Clara
A2-BUF					1	x STAFF-NAS-CLINT
A2-BUF					1	x Staff-Nas-Dona
A2-BUF					1	x Staff-Nas-Graeme
A2-BUF					1	x Staff-OWS-Russel
A2-BUF					1	x Staff-Sefofane-Witness
A2-BUF	14:40	OMD	14:50	HND	1	x NAS-Staff-Joanne
A2-BUF					1	x Staff-Nas-Clara
A2-BUF					1	x STAFF-NAS-CLINT
A2-BUF					1	x Staff-Nas-Dona
A2-BUF					1	x Staff-Nas-Graeme
A2-BUF					1	x Staff-OWS-Russel
A2-BUF	15:00	HND	15:30	MUB	1	x NAS-Staff-Joanne
A2-BUF					1	x NAS-Staff-July
A2-BUF					1	x Staff-Nas-Clara
A2-BUF					1	x STAFF-NAS-CLINT
A2-BUF					1	x Staff-Nas-Dona
A2-BUF					1	x Staff-Nas-Graeme
A2-BUF					1	x Staff-OWS-Russel

Aircraft	*Etd*	*From*	*Eta*	*To*	*Pax*	*Group*
A2-EGL	10:15	ABU	10:40	MUB	7	x Keshavjee Party SOLE 208/2
A2-EGL					1	x Staff-Sefofane-Faadhil
A2-EGL	11:20	MUB	11:55	VUM	1	x Saff-Ows-Agatha
A2-EGL					1	x Staff-Ows-Bonolo
A2-EGL					1	x Staff-OWS-Joel
A2-EGL					1	x Staff-OWS-Keleemetse
A2-EGL					1	x Staff-Ows-Keolebogile
A2-EGL					1	x Staff-Ows-Kgakgamatso
A2-EGL					1	x Staff-OWS-Lalu
A2-EGL					1	x Staff-Ows-Lesego
A2-EGL					1	x Staff-Ows-Ortell
A2-EGL					1	x Staff-Ows-Sylvia
A2-EGL					1	x Staff-Ows-Tshubugo
A2-EGL					1	x Staff-Sefofane-Faadhil
A2-EGL	12:05	VUM	12:15	SRA	1	x Staff-OWS-Baeti
A2-EGL					1	x Staff-OWS-Bob
A2-EGL					1	x Staff-OWS-Changi
A2-EGL					1	x Staff-Ows-Felicia
A2-EGL					1	x Staff-OWS-Gaba
A2-EGL					1	x Staff-OWS-Johannes
A2-EGL					1	x Staff-OWS-John
A2-EGL					1	x Staff-OWS-John
A2-EGL					1	x Staff-OWS-Kelly
A2-EGL					1	x Staff-OWS-Koi
A2-EGL					1	x Staff-Ows-Nana
A2-EGL					1	x Staff-Sefofane-Faadhil
A2-EGL	12:25	SRA	12:35	OMD	1	x Staff-OWS-Gale
A2-EGL					1	x Staff-OWS-Kelly
A2-EGL					1	x Staff-OWS-Letty

Aircraft	*Etd*	*From*	*Eta*	*To*	*Pax*	*Group*
A2-EGL					1	x Staff-OWS-Mokopi
A2-EGL					1	x Staff-OWS-Phetso
A2-EGL					1	x Staff-OWS-Tshotlego
A2-EGL					1	x Staff-Sefofane-Faadhil
A2-EGL	12:45	OMD	12:55	SRA	1	x Staff-OWS-Bashie
A2-EGL					1	x Staff-Ows-Boitumelo
A2-EGL					1	x Staff-OWS-Celia
A2-EGL					1	x Staff-OWS-Mokopi
A2-EGL					1	x Staff-Ows-Olatotswe
A2-EGL					1	x Staff-OWS-Smiley
A2-EGL					1	x Staff-Sefofane-Faadhil
A2-EGL	13:05	SRA	13:15	VUM	1	x Staff-Ows-Bole
A2-EGL					1	x Staff-OWS-Glorius
A2-EGL					1	x Staff-OWS-Keleemetse
A2-EGL					1	x Staff-OWS-KK
A2-EGL					1	x Staff-Ows-Lebasho
A2-EGL					1	x Staff-OWS-Marriam
A2-EGL					1	x Staff-OWS-Motsumi
A2-EGL					1	x Staff-OWS-Motty
A2-EGL					1	x Staff-OWS-OB
A2-EGL					1	x Staff-OWS-Sadek
A2-EGL					1	x Staff-OWS-Tshenyego M
A2-EGL					1	x Staff-Sefofane-Faadhil
A2-EGL	13:25	VUM	14:00	MUB	1	x Staff-Ows-Disho
A2-EGL					1	x Staff-ows-Dolly
A2-EGL					1	x Staff-Ows-Lebasho
A2-EGL					1	x Staff-Sefofane-Faadhil

Aircraft	*Etd*	*From*	*Eta*	*To*	*Pax*	*Group*
A2-GNU	10:35	MUB	11:50	VFA	1	x Staff-OWS-Thuto
A2-GNU	12:30	VFA	13:15	MLO	12	x BOX#69(05/20)08
A2-GNU	13:35	MLO	14:20	VFA	12	x BOX#67(05/20)2008
A2-GNU	15:00	VFA	15:20	BBK	0	Empty

Aircraft	*Etd*	*From*	*Eta*	*To*	*Pax*	*Group*
A2-FOX	10:15	MUB	11:00	XOR	0	Empty
A2-FOX	11:10	XOR	11:35	XIG	1	x Z1-11(08)Guide-Francis
A2-FOX					1	x Z1-11(08)Macgregor
A2-FOX					1	x Z1-11(08)McLean
A2-FOX	11:45	XIG	11:50	MOM	0	Empty
A2-FOX	12:00	MOM	13:30	BBK	2	x Levinson 7nt

Aircraft	*Etd*	*From*	*Eta*	*To*	*Pax*	*Group*
A2-JKL	9:50	BBK	11:15	VUM	0	Empty
A2-JKL	11:25	VUM	12:00	CBE	2	x Stewart 7nt
A2-JKL	12:10	CBE	13:05	BBK	2	x Bethoney 7nt
A2-JKL					2	x Yoshimoto 7nt
A2-JKL	13:45	BBK	14:10	LVI	2	x Bethoney 7nt
A2-JKL					2	x Levinson 7nt
A2-JKL	14:40	LVI	15:05	BBK	0	Empty

Aircraft	*Etd*	*From*	*Eta*	*To*	*Pax*	*Group*
A2-LEO	8:20	MUB	8:45	ABU	1	x Staff-Sefofane-Patrick
A2-LEO	9:05	ABU	9:30	MUB	6	x Keshavjee Party SOLE
A2-LEO					1	x Staff-Sefofane-Patrick
A2-LEO	13:10	MUB	13:35	MOM	2	x Levy
A2-LEO					2	x MILLER 7nt
A2-LEO					2	x Shields

Aircraft	*Etd*	*From*	*Eta*	*To*	*Pax*	*Group*
A2-LEO					1	x Staff-OWS-Landi
A2-LEO					1	x Staff-OWS-Rob
A2-LEO					1	x Staff-Sefofane-Ollie
A2-LEO	13:45	MOM	13:55	HND	2	x MILLER 7nt
A2-LEO					2	x Shields
A2-LEO					1	x Staff-Sefofane-Ollie
A2-LEO	14:05	HND	14:15	VUM	2	x MILLER 7nt
A2-LEO					1	x Staff-Sefofane-Ollie
A2-LEO					4	x World Jouney Edu/1
A2-LEO					4	x World Jouneys Edu/2
A2-LEO	14:25	VUM	14:35	HND	1	x Staff-Sefofane-Ollie

Aircraft	*Etd*	*From*	*Eta*	*To*	*Pax*	*Group*
A2-OWL	8:15	VUM	9:00	MUB	3	x McLachlan
A2-OWL	10:25	MUB	10:55	SHN	1	x Staff-OWS-Taps
A2-OWL					1	x Staff-OWS-TK
A2-OWL	11:05	SHN	11:20	MOM	2	x Michaela
A2-OWL					1	x Staff-OWS-Taps
A2-OWL					1	x Staff-OWS-TK
A2-OWL	11:30	MOM	11:35	JAO	2	x Shappell 7nt
A2-OWL					1	x Staff-OWS-Russel
A2-OWL	11:45	JAO	12:25	CBE	2	x Rifkin
A2-OWL					2	x Shappell 7nt
A2-OWL	12:35	CBE	13:25	ABU	2	x Davis
A2-OWL					2	x Koeppel
A2-OWL	13:35	ABU	14:10	MUB	2	x Koeppel
A2-OWL	15:45	MUB	16:35	CBE	2	x Long
A2-OWL					1	x Staff-Sable-Cathy

Aircraft	*Etd*	*From*	*Eta*	*To*	*Pax*	*Group*
A2-OWL					1	x Staff-Sable-Rocky
A2-OWL	16:45	CBE	16:50	SLI	0	Empty

Aircraft	*Etd*	*From*	*Eta*	*To*	*Pax*	*Group*
A2-ZEB	10:40	LVI	11:00	BBK	4	x Beckerman
A2-ZEB					2	x Davis
A2-ZEB					2	x Thomas
A2-ZEB	11:40	BBK	12:20	CBE	4	x Beckerman
A2-ZEB					2	x Davis
A2-ZEB					1	x Staff-Linyanti
A2-ZEB					1	x Staff-Sefofane-Tumo
A2-ZEB					2	x Thomas
A2-ZEB	12:30	CBE	12:35	SLI	2	x Bartenev
A2-ZEB					1	x Bartenev Extra Seat
A2-ZEB					2	x Carthy
A2-ZEB					2	x Christenson
A2-ZEB					1	x Staff-Linyanti Explo-Fostr
A2-ZEB					1	x Staff-Sefofane-Tumo
A2-ZEB					2	x Thomas
A2-ZEB	12:45	SLI	13:05	VUM	2	x Bartenev
A2-ZEB					1	x Bartenev Extra Seat
A2-ZEB					2	x Christenson
A2-ZEB					2	x Pradere
A2-ZEB					1	x Staff-Sefofane-Tumo
A2-ZEB	13:15	VUM	13:20	OMD	2	x Bell
A2-ZEB					2	x Christenson
A2-ZEB					2	x Pawliczek
A2-ZEB					2	x Pradere
A2-ZEB					1	x Staff-OWS-Camilla

Aircraft	***Etd***	***From***	***Eta***	***To***	***Pax***	***Group***
A2-ZEB					1	x Staff-Sefofane-Tumo
A2-ZEB	13:30	OMD	13:35	MOM	2	x Bell
A2-ZEB					2	x Pawliczek
A2-ZEB					2	x Pradere
A2-ZEB					1	x Staff-OWS-Camilla
A2-ZEB					1	x Staff-Sefofane-Tumo
A2-ZEB	13:45	MOM	13:50	XIG	2	x Bell
A2-ZEB					2	x Gilels 7nt
A2-ZEB					2	x Pradere
A2-ZEB					1	x Staff-Sefofane-Tumo
A2-ZEB					2	x Steel
A2-ZEB					2	x Word
A2-ZEB	14:00	XIG	14:25	MUB	2	x Bell
A2-ZEB					2	x Gilels 7nt
A2-ZEB					1	x Staff-Sefofane-Tumo
A2-ZEB					2	x Steel
A2-ZEB					2	x Word

Appendix XII: The booking list of 175 flight bookings

From	*To*	*Pax*	*Etd*	*Ltd*	*Eta*	*Lta*	*Group Name*	*BI*
XOR	XIG	1	1100	1125	1435	1500	Z1-11(08)Macgregor	1
BBK	CBE	2	1330	1425	1400	1455	Z3-10(08)Kauth	2
BBK	CBE	1	1330	1425	1400	1455	Z3-10(08)Guide Richard	3
KWD	SWI	5	1100	1140	1420	1500	BOX 71(08)/2	4
CBE	MUB	1	700	745	715	800	Boehmke/2	5
CBE	VUM	2	1100	1135	1455	1530	Bartenev	6
MUB	JAO	4	1245	1320	1455	1530	Merrill 7nt	7
CTB	VUM	2	1100	1125	1505	1530	Holden	8
CBE	BBK	2	1100	1200	1430	1530	Bethoney 7nt	9
VUM	MUB	2	1250	1335	1315	1400	Bell	10
MOM	CBE	2	1100	1135	1455	1530	Shappell 7nt	11
CBE	TSO	2	1100	1230	1400	1530	Luecke	12
JAO	CBE	2	1100	1145	1445	1530	Rifkin	13
MUB	TSO	2	1250	1350	1430	1530	Farley	14
MUB	CTB	2	1250	1310	1510	1530	Sollek & McNally	15
SRA	VUM	1	600	1800	600	1800	Staff-OWS-Glorius	16
SRA	OMD	1	600	1800	600	1800	Staff-OWS-Gale	17
OMD	SRA	1	600	1800	600	1800	Staff-Ows-Boitumelo	18
OMD	SRA	1	600	1800	600	1800	Staff-Ows-Olatotswe	19

From	*To*	*Pax*	*Etd*	*Ltd*	*Eta*	*Lta*	*Group Name*	*BI*
SHN	MOM	2	1100	1430	1200	1530	Michaela	20
MUB	VUM	1	600	1800	600	1800	Staff-Ows-Sentle	21
MUB	MOM	1	600	1800	600	1800	Staff-OWS-Landi	22
BBK	SLI	1	600	1800	600	1800	Staff-Linyanti Explo-Foster	23
CBE	MUB	1	600	1800	600	1800	Staff-Sable-OB	24
SWI	MUB	1	600	1800	600	1800	Staff-OWS-Lindi	25
MUB	VUM	1	600	1800	600	1800	Staff-OWS-Moa	26
JAO	MUB	1	600	1800	600	1800	STAFF-NAS-CLINT	27
JAO	MUB	1	600	1800	600	1800	Staff-Nas-Graeme	28
MUB	SWI	1	600	1800	600	1800	Staff-OWS-KG	29
XOR	XIG	1	1100	1125	1435	1500	Z1-11(08)McLean	30
BBK	KWD	12	1000	1100	1400	1500	BOX 73(08)/1	31
SWI	BBK	12	700	820	800	920	BOX 69(08)/1	32
CTB	JAO	2	1100	1125	1505	1530	Thomas & Lorenz	33
MOM	MUB	2	1300	1335	1325	1400	Steel	34
VUM	TSO	2	1100	1240	1350	1530	Trimble 7nt	35
MOM	MUB	2	1250	1335	1325	1400	Gilels 7nt	36
HND	VUM	4	1100	1115	1515	1530	World Jouneys Edu/2	37
LVI	BBK	2	1000	1025	1400	1425	Thomas	38
BBK	CBE	4	1100	1200	1430	1530	Beckerman	39
MUB	XIG	4	1350	1420	1500	1530	Wright	40
SLI	XIG	2	1100	1140	1450	1530	Pradere	41
XIG	BBK	2	1010	1200	1050	1230	Harris	42
ABU	MUB	6	1100	1135	1455	1530	Keshavjee Party SOLE 208/1	43
MUB	OMD	2	1250	1330	1450	1530	Mahul Shah	44
BBK	LVI	2	1310	1335	1505	1530	Levinson 7nt	45

From	*To*	*Pax*	*Etd*	*Ltd*	*Eta*	*Lta*	*Group Name*	*BI*
SRA	VUM	1	600	1800	600	1800	Staff-Ows-Bole	46
SRA	VUM	1	600	1800	600	1800	Staff-OWS-Motsumi	47
SRA	VUM	1	600	1800	600	1800	Staff-OWS-Motty	48
SRA	VUM	1	600	1800	600	1800	Staff-OWS-OB	49
SRA	VUM	1	600	1800	600	1800	Staff-OWS-Tshenyego M	50
VUM	SRA	1	600	1800	600	1800	Staff-OWS-John	51
SRA	OMD	1	600	1800	600	1800	Staff-OWS-Phetso	52
OMD	SRA	1	600	1800	600	1800	Staff-OWS-Smiley	53
OMD	SRA	1	600	1800	600	1800	Staff-OWS-Mokopi	54
SRA	VUM	1	600	1800	600	1800	Staff-Ows-Lebasho	55
VUM	MUB	1	600	1800	600	1800	Staff-Ows-Lebasho	56
SRA	VUM	1	600	1800	600	1800	Staff-OWS-Sadek	57
MUB	VUM	1	600	1800	600	1800	Staff-OWS-Lalu	58
XOR	XIG	1	1100	1125	1435	1500	Z1-11(08)Guide-Francis	59
SWI	BBK	5	700	820	800	920	BOX 69(08)/2	60
CTB	MUB	2	1250	1320	1330	1400	Spilsbury	61
XIG	MUB	2	805	835	830	900	Marrero	62
VUM	CBE	2	1100	1135	1455	1530	Stewart 7nt	63
JAO	CTB	4	1100	1125	1505	1530	Higbee Stacy	64
CBE	MUB	2	1310	1400	1350	1440	Koeppel	65
VUM	MOM	2	1100	1115	1515	1530	Pawliczek	66
CBE	OMD	2	1100	1135	1455	1530	Christenson	67
HND	VUM	4	1100	1115	1515	1530	World Jouney Edu/1	68
JAO	MUB	3	800	835	825	900	Auster 7nt	69
CBE	SLI	2	1100	1430	1455	1530	Carthy	70
MUB	MOM	2	1250	1325	1455	1530	Levy	71
MUB	HND	2	1230	1310	1450	1530	Shields	72
MUB	CTB	2	1250	1310	1510	1530	Freeman Kalm	73

From	*To*	*Pax*	*Etd*	*Ltd*	*Eta*	*Lta*	*Group Name*	*BI*
MUB	TSO	0	1250	1335	1330	1525	Staff-OWS-Changi	74
VUM	SRA	1	600	1800	600	1800	Staff-OWS-Keleemetse	75
SRA	VUM	1	600	1800	600	1800	Staff-OWS-Mokopi	76
SRA	OMD	1	600	1800	600	600	Staff-OWS-Marriam	77
SRA	VUM	1	600	1800	600	1800	Staff-OWS-KK	78
SRA	VUM	1	600	1800	600	1800	Staff-Ows-Bonolo	79
MUB	VUM	1	600	1800	600	1800	Staff-Ows-Gaositege	80
MUB	VUM	1	600	1800	600	1800	Davis	81
LVI	BBK	2	1000	1025	1505	1530	Staff-Ows-Letty	82
MUB	MOM	1	600	1800	600	1800	Staff-OWS-Rob	83
MUB	MOM	1	600	1800	600	1800	BOX (05/20)08	84
VFA	MLO	12	1215	1305	1230	1315	Staff-Sefofane-Ollie	85
MUB	VUM	1	600	1800	600	1800	Staff-Sefofane-Faadhil	86
MUB	MUB	1	600	1800	600	1800	Staff-OWS-Gloria	87
VUM	SRA	1	600	1800	600	1800	Staff-OWS-Baeti	88
VUM	SRA	1	600	1800	600	1800	Staff-Ows-Felicia	89
VUM	SRA	1	600	1800	600	1800	nas-staff-keowetse	90
JAO	MUB	1	600	1800	600	1800	NAS-Staff-Isaac	91
JAO	MUB	1	600	1800	600	1800	Staff-Ows-Disho	92
VUM	MUB	1	600	1800	600	1800	Staff-Sable-Rocky	93
MUB	CBE	1	600	1800	600	1800	Staff-Sefofane-Lekani	94
BBK	MUB	1	600	1800	600	1800	Staff-Great Explo-OT	95
MUB	XIG	1	600	1800	600	1800	Staff-Ows-Tirelo	96
MUB	SWI	1	600	1800	600	1800	Staff-OWS-KK	97
MUB	SWI	1	600	1800	600	1800	Staff-Sable-Raymond	98
CBE	MUB	1	1100	1800	1430	1800	Staff-OWS-Bob	99
VUM	SRA	1	600	1800	600	1800	Staff-Ows-Nana	100
VUM	SRA	1	600	1800	600	1800	Staff-OWS-Johannes	101

From	*To*	*Pax*	*Etd*	*Ltd*	*Eta*	*Lta*	*Group Name*	*BI*
VUM	SRA	1	600	1800	600	1800	Staff-OWS-Letty	102
SRA	OMD	1	600	1430	1200	1530	Staff-OWS-Keleemetse	103
MUB	VUM	1	600	1800	600	1800	Tools	104
JAO	OMD	1	600	1800	600	1800	BOX 73(08)/2	105
BBK	KWD	2	1000	1100	1400	1500	BOX 71(08)/1	106
KWD	SWI	12	1100	1140	1420	1500	Boehmke/1	107
CBE	MUB	3	700	745	715	800	Yoshimoto 7nt	108
CBE	BBK	2	1100	1200	1430	1530	Bethoney 7nt	109
BBK	LVI	2	1230	1255	1505	1530	McLachlan	110
VUM	MUB	3	750	835	815	900	Stahl	111
SLI	BBK	4	1100	1200	1430	1530	Thomas	112
BBK	SLI	2	1100	1200	1430	1530	Beckerman	113
LVI	BBK	4	1000	1025	1400	1425	Angel Ortega	114
OMD	SLI	2	1100	1120	1450	1530	Word	115
MOM	MUB	2	1300	1335	1325	1400	Keshavjee Party SOLE 208/2	116
ABU	MUB	7	1100	1135	1455	1530	Long	117
MUB	CBE	2	1550	1630	1700	1750	MILLER 7nt	118
MUB	VUM	2	1250	1335	1445	1530	Levinson 7nt	119
MOM	BBK	2	1100	1240	1350	1530	Bartenev Extra Seat	120
CBE	VUM	1	1100	1135	1455	1530	Staff-OWS-Kenny	121
VUM	SRA	1	600	1800	600	1800	Staff-OWS-John	122
VUM	SRA	1	600	1800	600	600	Staff-OWS-Tshotlego	123
SRA	OMD	1	600	1800	600	1800	Staff-OWS-Celia	124
OMD	SRA	1	600	1800	600	1800	Staff-OWS-Bashie	125
OMD	SRA	1	600	1800	600	1800	Staff-OWS-Martha	126
MUB	VUM	1	600	1800	600	1800	Staff-Ows-Ortell	127
MUB	VUM	1	600	1800	600	1800	Staff-Ows-Tshubugo	128
MUB	VUM	1	600	1800	600	1800	Staff-Ows-Keolebogile	129

From	*To*	*Pax*	*Etd*	*Ltd*	*Eta*	*Lta*	*Group Name*	*BI*
MUB	VUM	1	600	1800	600	1800	Saff-Ows-Agatha	130
MUB	VUM	1	600	1800	600	1800	Staff-Ows-Kgakgamatso	131
MUB	VUM	1	600	1800	600	1800	Staff-Ows-Sylvia	132
MUB	VUM	1	600	1800	600	1800	Staff-Ows-Lesego	133
JAO	HND	0	1100	1055	1515	1530	Staff-OWS-Joel	134
MUB	VUM	1	600	1800	600	1800	Farley Extra Seat Luggage	135
MUB	VUM	1	600	1800	600	1800	Davis	136
MUB	TSO	1	1250	1335	1330	1525	Staff-OWS-Thuto	137
BBK	ABU	2	1100	1300	1330	1530	Staff-Flamingo-Lebopo	138
MUB	BBK	1	600	1800	600	1800	Staff-Sefofane-Warren	139
CTB	MUB	1	600	1800	600	1800	Staff-Sefofane-Tumo	140
MUB	CBE	1	600	1800	600	1800	Staff-Sable-Cathy	141
BBK	MUB	1	600	1800	600	1800	Staff-Flamingo-Simon	142
MUB	CBE	1	600	1140	600	1530	Staff-OWS-Martha	143
CTB	MUB	1	600	1800	600	1800	Staff-Flamingo-OT	144
MUB	VUM	1	600	1800	600	1800	Staff-Nas-Dona	145
CTB	MUB	1	600	1800	600	1800	Staff-Nas-Clara	146
JAO	MUB	1	600	1800	600	1800	Staff-OWS-Taps	147
JAO	MUB	1	600	1800	600	1800	Staff-OWS-Russel	148
MUB	MOM	1	600	1800	600	1800	Staff-OWS-Koi	149
JAO	MUB	1	600	1800	600	1800	Staff-ows-Dolly	150
VUM	SRA	1	600	1800	600	1800	Staff-Ows-Letty	151
VUM	MUB	1	600	1800	600	1800	NAS-Staff-Joanne	152
MUB	MOM	1	600	1800	600	1800	NAS-Staff-Lebo	153
JAO	MUB	1	600	1800	600	1800	NAS-Staff-Mike	154
HND	MUB	1	600	1800	600	1800	Staff-Sable-Lapo	155
JAO	MUB	1	600	1800	600	1800	Kirby 7nt	156
MUB	CBE	1	600	1800	600	1800	Staff-Ows-Bole	157

From	*To*	*Pax*	*Etd*	*Ltd*	*Eta*	*Lta*	*Group Name*	*BI*
CTB	JAO	3	1100	1125	1505	1530	Staff-OWS-Camilla	158
MUB	VUM	1	600	1800	600	1800	BOX 67(05/20)2008	159
VUM	MOM	1	1100	1800	1430	1800	Staff-Sefofane-Modiri	160
MLO	VFA	12	1335	1420	1345	1430	Staff-Sefofane-Witness	161
CBE	MUB	1	600	1800	600	1800	Staff-Sefofane-Patrick	162
MUB	VUM	1	600	1800	1200	1800	NAS-Staff-July	163
MUB	MUB	1	600	1800	600	1800	Staff-OWS-Sam	164
HND	MUB	1	600	1800	600	1800	Staff-OWS-KP	165
SWI	MUB	1	600	1800	600	1800	Staff-Sable-Lee	166
SWI	MUB	1	600	1800	600	1800	Staff-GREAT Explo-Goweditswe	167
CBE	MUB	1	600	1800	600	1800	Staff-Ows-Keoikantse	168
MUB	XIG	1	600	1800	600	1800	Staff-OWS-TK	169
MUB	VUM	1	600	1800	600	1800	Staff-OWS-Gaba	170
MUB	MOM	1	600	1800	600	1800	Staff-OWS-Kelly	171
VUM	SRA	1	600	1800	600	1800	Staff-OWS-Russel	172
SRA	OMD	1	600	1430	1200	1530	Staff-OWS-Chris Bates	173
MOM	JAO	1	600	1800	600	1800	Nas-Staff-Sam	174
JAO	OMD	1	600	1800	600	1800	NAS-Staff-Nelly	175

Appendix XIII: Matlab codes

```
clear global aircraft;
clear global demand;
clear global distances;
clear global pilot;
clear global locations;
clear global coords;
clear global flights;
clear global airflights;
clear global pilflights;
clear global cipdata;
clear global newcipdata;
tic;
t1 = cputime;
global aircraft;
global demand;
global distances;
global locations;
global pilot
global coords;
global flights;
global cipdata;
global newcipdata;

vers = '1.00 (2011-02-12)';

%% Stage 1 %% Reading and sorting information
    %%% Read Scheduling Parameters. Such as Demand, Locations,
    %%% Planes, etc...
    [demand, locations, aircraft, demand.details, pilot] = Treaddata;
    %%% Parse databases. (generate distance table, sort data,
    %%% assign index numbers
    [distances,coords,demand,aircraft,pilot] = Tparser;

% Stage 2 %% Calculation of CIP variables, constraints and coeffients.
%% Formation of possible flight leg groupings.
    flights = zeros(size(demand.data,1),5000); %%pre-allocated for speed
    presolver1;
%% Creates possible individual flights with cost ans timing recorded.
    singles;
    linkstart = size(cipdata.selected,1)+1;
%% Creates links flights into possible pairs.
```

```
    linker;
%% Creates possible chains.
    chainer(linkstart);
    clear flights;
    display('~~ Creating CIP Data.');
% Creates ten minute time blocks.
    Tnodes;
    newtimes;
    disp(strcat('~~ Number of Variables: '...
    ,num2str(size(newcipdata.selected,1))));
%Creates the empty flights
    Tsource;
    Tgroundnodes;
    disp(strcat('~~ Total Number of Variables: '...
    ,num2str(size(newcipdata.selected,1))));
%% Record variables for solver.
    Trecordconnector;
% Stage 3 %% Formation of Zimpl Files
    TSefzplcreate;
    %% Saves a Matlab MAT file for all the variables in the workspace.
    save Tphoenix;
    %% Footer of Phoenix Primo
    display('.....................................');
    display('// Completed.');
    %% Stats
    t2 = cputime - t1;
    t3 = toc;
    disp(strcat('   CPU Time: ',num2str(t2),' seconds'));
    disp(strcat('   Elasped Time: ',num2str(t3),' seconds'));


function [demandout,locationsout,aircraftout,demanddetailsout,...
pilotout] = Treaddata
%READDATA Reads flight scheduling data and outputs simple matrices
%   Reads structed csv files containing data and outputs a matrix for each file.

%% Should have a GUI for selecting files.
answer = {'files/input/demand.csv','files/input/locations.csv','files/input
/aircraft.csv','files/input/demand-details.csv','files/input/pilot.csv'};
demandout = importdata(answer{1});
locationsout = importdata(answer{2});
aircraftout = importdata(answer{3});
demanddetailsout = importdata(answer{4});
```

```
pilotout = importdata(answer{5});
display('** Loaded databases from file.');
end

function [distances0,coords0,demand0,aircraft0,pilot0] = Tparser
%PARSER Generate distance table, sort data, assign index numbers
%   Basically this functions purpose is to organise the data into a more
%   useful structure for the rest of the program

global demand;
global locations;
global aircraft;
global pilot;

locsize = size(locations.data,1);
j = 1;
dms = sparse(locsize*2,3);
for i=1:1:locsize
    dms(j,1) = locations.data(i,1);
    dms(j,2) = locations.data(i,2);
    dms(j,3) = locations.data(i,3);
    dms(j+1,1) = locations.data(i,4);
    dms(j+1,2) = locations.data(i,5);
    dms(j+1,3) = locations.data(i,6);
    j = j + 2;
end

format long g;
angles = dms2deg(dms);
m = size(angles,1);
coords0 = sparse(m/2,2);
k=1;
for i=1:1:m
    if rem(i,2) == 0
        coords0(k,2) = angles(i,1);
        k = k+1;
    else
        coords0(k,1) = angles(i,1);
    end
end
distances0 = sparse(locsize,locsize);
for a=1:1:locsize
    for b=1:1:locsize
```

```
        distances0(a,b) = round(deg2km(distanc(coords0(a,2),
         coords0(a,1),coords0(b,2), coords0(b,1))));
    end
end
display('** Created distances table.');
disp(strcat('   Number of Locations: ',num2str(locsize)));
demand0= demand;
demsize = size(demand.data,1);
paxnum = 0;
for k=1:1:demsize
    demand0.data(k,7) = k;
    for m=1:1:locsize
        if (strcmp(demand0.textdata(k+1,1),locations.textdata(m+1,1)))
            demand0.data(k,8) = m;%% start location code index
        end
    end
    for n=1:1:locsize
        if (strcmp(demand0.textdata(k+1,2),locations.textdata(n+1,1)))
            demand0.data(k,9) = n; %% end location code index
        end
    end
    paxnum = paxnum + demand0.data(k,1);
end
display('** Converted location codes in demand database.');
display('** Enumarated bookings in demand database.');
disp(strcat('   Number of Bookings: ',num2str(demsize)));
disp(strcat('   Number of Passengers: ',num2str(paxnum)));
demand0.orgsorted = demand0.data;
demand0.data = sortrows(demand0.data,[8 9 1 2 3 4 5]);
display('** Rearranged demand database.');

aircraft0 = aircraft;
aircraft0.types(1,1:10) = aircraft0.data(1,1:10); %% specs of aircraft
aircraft0.types(1,11) = 1; %% number of type of aircraft
aircraft0.data(1,11) = 1;
infostring = strcat('   Aircraft Types: (1) ',aircraft0.textdata(2,2));
for c=2:1:size(aircraft0.data,1)
    found = 0;
    for d=1:1:size(aircraft0.types,1)
        if (aircraft0.data(c,1:10)==aircraft0.types(d,1:10))
            aircraft0.types(d,11) = aircraft0.types(d,11) + 1;
            found = 1;
            aircraft0.data(c,11) = d;
```

```
        end
    end
    if (found == 0)
        si = size(aircraft0.types,1)+1;
        aircraft0.types(si,1:10) = aircraft0.data(c,1:10);
        aircraft0.types(si,11) = 1;
        aircraft0.data(c,11) = si;
        infostring = strcat(infostring,'  (',num2str(si),') ',
        aircraft0.textdata(c+1,2));
    end
end
slowest = 1;
fastest = 1;
    for pl=2:1:size(aircraft0.types,1)
        if (aircraft0.types(pl,1) < aircraft0.types(slowest,1))
           slowest = pl;
        end;
        if (aircraft0.types(pl,1) > aircraft0.types(fastest,1))
           fastest = pl;
        end;
    end;

    codb = 0;
for lp=1:1:size(demand0.data,1)
    if (demand0.data(lp,4) ~= 600)
    timemin = (distances0(demand0.data(lp,8),demand0.data(lp,9))...
    /aircraft0.types(slowest,1))*60;
    timeadded = timeadd(timemin,demand0.data(lp,3));
    if (demand0.data(lp,4) > timeadded)
        codb = codb + 1;
        timemin2 = (distances0(demand0.data(lp,8),demand0.data(lp,9))...
        /aircraft0.types(fastest,1))*60;
        timeadded2 = timeadd(timemin2,demand0.data(lp,3));
        demand0.data(lp,3) = demand0.data(lp,5);
        demand0.data(lp,4) = timeadded2; %% removed: ten minute window.
    end
    end
end
    for t = 1:1:size(aircraft0.types,1)
        i  =0;
        %aircraft.loc(t).data = zeros(size(aircraft.textdata,1),13);
        for c = 1:1:size(aircraft.data,1)
            if (aircraft.data(c,1:10) == aircraft0.types(t,1:10))
```

```
i = i+1;
aircraft0.heuristic(t).data(i,1) = c;% aircraft index;
aircraft0.heuristic(t).data(i,2) = t;% aircraft type
aircraft0.heuristic(t).data(i,4:13) = aircraft.data(c,1:10);
              for m = 1:1:size(locations.data,1)
if (strcmp(aircraft.textdata(c+1,3),locations.textdata(m+1,1)))
    aircraft0.heuristic(t).data(i,3) = m;%aircraft location
end
              end
            end
        end
    end

pilot0 = pilot;
pilot0.info(t).data = zeros(0,0);
for p = 1:1: size(pilot.data,1)
    for a = 3 : size(pilot.textdata,2)-2
        for t = 1:1:size(aircraft0.types,1)
            if strcmp(pilot.textdata(p+1,a),
            aircraft.textdata(aircraft0.heuristic(t).data(1,1)+1,2))
j = size(pilot0.info(t).data,1) +1 ;
pilot0.info(t).data(j,1) = p;
for n = 1:1:size(locations.data,1)
    if strcmp(pilot.textdata(p+1,2),locations.textdata(n+1,1))
        pilot0.info(t).data(j,2) = n;
    end
end
pilot0.info(t).data(j,3) = pilot.data(p,1);
pilot0.info(t).data(j,4) = pilot.data(p,2);
            end
        end
    end
end

demand0.sorted = sortrows(demand0.data,[7]);
disp(strcat('   Number of Bookings Adjusted due to Timing Conflicts: '
,num2str(codb)));display('** Analysed Aircraft.');
disp(strcat('   Number of Types of Aircraft: ',...
num2str(size(aircraft0.types,1))));
disp(strcat('   Number of Aircraft in Fleet: ',...
num2str(size(aircraft0.data,1))));
disp(infostring{1});
end
```

```
function deg=dms2deg(dms)
% DMS2DEG  Converts degrees-minutes-seconds to decimal degrees.
%   Vectorized.
% Version: 12 Mar 00
% Useage:  deg=dms2deg(dms)
% Input:   dms - [d m s] array of angles in deg-min-sec, where
%d = vector of degrees
%m = vector of minutes
%s = vector of seconds
% Output:  deg - vector of angles in decimal degrees
d=dms(:,1);
m=dms(:,2);
s=dms(:,3);
deg=abs(d)+abs(m)./60+abs(s)./3600;
ind=(d<0 | m<0 | s<0);
deg(ind)=-deg(ind);

function km = deg2km(deg,sphere)
%DEG2KM Convert distance from degrees to kilometers
%
%   KM = DEG2KM(DEG) converts distances from degrees to kilometers as
%   measured along a great circle on a sphere with a radius of 6371 km, the
%   mean radius of the Earth.
%
%   KM = DEG2KM(DEG,RADIUS) converts distances from degrees to kilometers
%   as measured along a great circle on a sphere having the specified
%   radius. RADIUS must be in units of kilometers.
%
%   KM = DEG2KM(DEG,SPHERE) converts distances from degrees to kilometers,
%   as measured along a great circle on a sphere approximating an object in
%   the Solar System.  SPHERE may be one of the following strings: 'sun',
%   'moon', 'mercury', 'venus', 'earth', 'mars', 'jupiter', 'saturn',
%   'uranus', 'neptune', or 'pluto', and is case-insensitive.
%
%   See also DEG2NM, DEG2RAD, DEG2SM, KM2DEG.

% Copyright 1996-2007 The MathWorks, Inc.
% $Revision: 1.10.4.3 $  $Date: 2007/03/27 19:11:50 $
rad = deg2rad(deg);

if nargin == 1
    km = rad2km(rad);
```

```
else
    km = rad2km(rad,sphere);
end


function d = distanc(aa, bb, cc, dd)
% DISTANCE - computes Euclidean distance matrix
%
% E = distance(A,B)
%
%    A - (DxM) matrix
%    B - (DxN) matrix
%
% Returns:
%    E - (MxN) Euclidean distances between vectors in A and B
%
%
% Description :
%    This fully vectorized (VERY FAST!) m-file computes the
%    Euclidean distance between two vectors by:
%
% ||A-B|| = sqrt ( ||A||^2 + ||B||^2 - 2*A.B )
%
% Example :
%    A = rand(400,100); B = rand(400,200);
%    d = distance(A,B);

% Author   : Roland Bunschoten
%            University of Amsterdam
%            Intelligent Autonomous Systems (IAS) group
%            Kruislaan 403  1098 SJ Amsterdam
%            tel.(+31)20-5257524
%            bunschot@wins.uva.nl
% Last Rev : Oct 29 16:35:48 MET DST 1999
% Tested   : PC Matlab v5.2 and Solaris Matlab v5.3
% Thanx    : Nikos Vlassis

% Copyright notice: You are free to modify, extend and distribute
%    this code granted that the author of the original code is
%    mentioned as the original author of the code.


d = sqrt((aa-cc)^2 + (bb-dd)^2);
```

```
function [output] = timeadd(min,t24)
%TIMEADD Adds minutes to a 2400 hour timestamp
    hr = floor(min/60);
    mins = min - (hr*60);
    minutest24 = t24 - (floor(t24/100)*100);
    if (minutest24 + mins >= 60)
        minout = minutest24 + mins - 60;
        hoursout = (floor(t24/100)*100) + hr*100 + 100;
        output = round(hoursout+minout);
    else
        hoursout = (floor(t24/100)*100) + hr*100;
        output = round(hoursout+minutest24+mins);
    end;
end

function presolver1
%PRESOLVER Heuristic for joining almost identical flights
global demand;
global aircraft;
global distances;
for t=1:1:size(aircraft.types,1)
    ended = 0;
    i = 1;
    extraspace = zeros(size(demand.data,1),25);
    temp = [demand.data extraspace];
    while (ended ~=1)
        k1 = size(temp,1);
        if (temp(i,1) > aircraft.types(t,4))
            temp(i,:) = [];
        else
            i = i + 1;
        end;
        if (k1 == i)
            ended = 1;
        end;
    end;
    ended = 0;
    i = 1;
    while (ended ~= 1)
        k = size(temp,1);
        if (i ~= k)
            if (temp(i,8) == temp(i,9)) || (temp(i,1) == 0)
```

```
temp(i,:) = [];
            else
if ((temp(i,8) ~= temp(i+1,8)) || (temp(i,9) ~= temp(i+1,9)))
    i = i + 1;
else
    if (temp(i,8) == temp(i+1,8))...
            && (temp(i,9) == temp(i+1,9))...
            && (temp(i,1) + temp(i+1,1) <= aircraft.types(t,4))...
            && (temp(i,2) == temp(i+1,2))...
            && (temp(i,3) == temp(i+1,3))...
            && (temp(i,4) == temp(i+1,4))...
            && (temp(i,5) == temp(i+1,5))
        temp(i,1) = temp(i,1) + temp(i+1,1);
        foundempty = 0;
        n = 10;
        while (foundempty == 0)
            if (temp(i,n) == 0)
foundempty = 1;
temp(i,n) = temp(i+1,7);
            end;
            n = n + 1;
        end;
        temp(i+1,:) = [];
    else
        if (temp(i,1) > aircraft.types(t,4))
            temp(i,:) = [];
        else
            if (temp(i,8) == temp(i+1,8))...
    && (temp(i,9) == temp(i+1,9))...
    && (temp(i,1) + temp(i+1,1) <= aircraft.types(t,4))...
    && (temp(i,2) == 600)...
    && (temp(i,3) == 1800)...
    && (temp(i,4) == 600)...
    && (temp(i,5) == 1800)
temp(i,1) = temp(i,1) + temp(i+1,1);
temp(i,2:5) = temp(i+1,2:5);
foundempty = 0;
n = 10;
while (foundempty == 0)
    if (temp(i,n) == 0)
        foundempty = 1;
        temp(i,n) = temp(i+1,7);
    end;
```

```
    n = n + 1;
end;
temp(i+1,:) = [];

            else
if (temp(i,8) == temp(i+1,8))...
        && (temp(i,9) == temp(i+1,9))...
        && (temp(i,1) + temp(i+1,1) <= aircraft.types(t,4))...
        && (temp(i+1,2) == 600)...
        && (temp(i+1,3) == 1800)...
        && (temp(i+1,4) == 600)...
        && (temp(i+1,5) == 1800)
    temp(i,1) = temp(i,1) + temp(i+1,1);
    foundempty = 0;
    n = 10;
    while (foundempty == 0)
        if (temp(i,n) == 0)
            foundempty = 1;
            temp(i,n) = temp(i+1,7);         end;
        n = n + 1;
    end;
    temp(i+1,:) = [];

else
    if (temp(i,8) == temp(i+1,8))... %%overlapping times
            && (temp(i,9) == temp(i+1,9))...
            && (temp(i,1) + temp(i+1,1) <= aircraft.types(t,4))...
            && (((temp(i+1,2) <= temp(i,3)) && (temp(i+1,2)...
             >= temp(i,2))) ||...
              ((temp(i+1,3) <= temp(i,3)) && (temp(i+1,3)...
               >= temp(i,2))))...
            && (((temp(i+1,4) <= temp(i,5)) && (temp(i+1,4)...
             >= temp(i,4))) ||...
              ((temp(i+1,5) <= temp(i,5)) && (temp(i+1,5)...
               >= temp(i,4))))
        if ((temp(i,8) ~= temp(i+1,8)) || (temp(i,9)...
         ~= temp(i+1,9)))
            i = i + 1;
        else
            timemins = (distances(temp(i,8),temp(i,9))...
            /aircraft.types(t,1))*60;
            if ( (min(temp(i,5),temp(i+1,5))) <= timeadd(timemins,...
            max(temp(i,2),temp(i+1,2))))...
```

```
            || ((max(temp(i,4),temp(i+1,4))) >= timeadd(timemins,...
            min(temp(i,3),temp(i+1,3))))
i = i + 1;
           else
if (temp(i,2)<=temp(i+1,2))
    temp(i,2) = temp(i+1,2);
end;
if (temp(i,3)>=temp(i+1,3))
    temp(i,3) = temp(i+1,3);
end;
if (temp(i,4)<=temp(i+1,4))
    temp(i,4) = temp(i+1,4);
end;
if (temp(i,5)>=temp(i+1,5))
    temp(i,5) = temp(i+1,5);
end;
temp(i,1) = temp(i,1) + temp(i+1,1);
foundempty = 0;
n = 10;
while (foundempty == 0)
    if (temp(i,n) == 0)
        foundempty = 1;
        temp(i,n) = temp(i+1,7);     end;
    n = n + 1;
end;
temp(i+1,:) = [];
            end;
        end;
    else
        i = i + 1;
    end;
end;
            end;
        end;
    end;
end;
            end;
        else
            ended = 1;
        end;
    end;
    temp = matshrink(temp);
    rightspace = zeros(size(temp,1),1);
```

```
    demand.heuristic(t).data = [temp rightspace];
end;
numre = 0;
for y=1:1:size(aircraft.types,1)
    numre = numre + (size(demand.data,1) -...
     size(demand.heuristic(t).data,1));
end;
display('** Applied First Heuristic.');
disp(strcat('   Number of Flights Removed: ',num2str(numre)));
end

function singles
%SINGLES Creates possible individual flights.
global demand;
global cipdata;
global flights;
global aircraft;
global distances;
varnum = 0;
for j=1:1:size(aircraft.types,1)
    for i=1:1:size(demand.heuristic(j).data,1)
        time1 = distances(demand.heuristic(j).data(i,8),...
        demand.heuristic(j).data(i,9))/aircraft.types(j,1);
        time2 = time1*60 + aircraft.types(j,8) + aircraft.types(j,9);
        cost = time1*(aircraft.types(j,5) + aircraft.types(j,6)) +...
         aircraft.types(j,10);
        varnum = varnum + 1;
        cipdata.time(varnum,j) = time2;
        cipdata.cost(varnum,1) = cost;
        ki = demand.heuristic(j).data(i,7);
        done = 0;
        y = 10;
        if (demand.heuristic(j).data(i,y) ~= 0)
            while (done == 0) &&...
             (y ~= size(demand.heuristic(j).data,2)+1)
if (demand.heuristic(j).data(i,y) == 0)
    done = 1;
else
    ki = [ki demand.heuristic(j).data(i,y)];
end;
y=y+1;
            end;
        end;
```

```
        cipdata.selected(varnum,1) = j;
        for cd=1:1:size(ki,2)
            flights(ki(1,cd),1) = flights(ki(1,cd),1) + 1;
            cipdata.selected(varnum,cd+1) = ki(1,cd);
            foundempty = 0;
            n = 2;
            while (foundempty == 0)
if (flights(ki(1,cd),n) == 0)
    flights(ki(1,cd),n) = varnum;
    foundempty = 1;
end;
n = n + 1;
            end;
        end;
    end;
end;
minutesinday = 720;
for i=1:1:size(aircraft.types,1)
    cipdata.times(i,1) = aircraft.types(i,11)*minutesinday;
end;
display('** Validating Single Flight Legs Completed.');
disp(strcat('   Number of Possible Single Flights: ',num2str(varnum)));
end

function linker
%LINKER Identifies possible flight groupings (Match-Maker)

global demand;
global aircraft;

numsel = 0;
numrel = 0;
for t=1:1:size(aircraft.types,1)
    for i=1:1:size(demand.heuristic(t).data,1)
        linkmids = zeros(50,3);
        qn = 0;
        for q=1:1:size(demand.heuristic(t).data,1)
   if (demand.heuristic(t).data(q,8) == demand.heuristic(t).data(i,8))...
    && (q ~= i)...
    && (demand.heuristic(t).data(q,8) ~= demand.heuristic(t).data(q,9))

qn = qn + 1;
linkmids(qn,1) = demand.heuristic(t).data(q,9);
```

```
linkmids(qn,2) = q;
            end;
        end;
        linkmids = matshrink(linkmids);
        chosen = zeros(50,3);
        ch = 0;
        if (qn ~= 0)
            for k=1:1:size(demand.heuristic(t).data,1)
if (demand.heuristic(t).data(k,8) == demand.heuristic(t).data(i,9))...
  && (demand.heuristic(t).data(k,8) ~= demand.heuristic(t).data(k,9))...
  && (i ~= k)...
  && (demand.heuristic(t).data(i,8) ~= demand.heuristic(t).data(k,9))
    for m=1:1:size(linkmids,1)
        if (demand.heuristic(t).data(k,9) == linkmids(m,1))
            ch = ch + 1;
            chosen(ch,1) = linkmids(m,2);
            chosen(ch,2) = k;
        end;
    end;
end;
            end;
        end;
        chosen = matshrink(chosen);
        if (size(chosen,1) ~= 0)
            for p=1:1:size(chosen,1)
maxpax1 = demand.heuristic(t).data(i,1) +...
 demand.heuristic(t).data(chosen(p,1),1);
maxpax2 = demand.heuristic(t).data(chosen(p,1),1) +...
 demand.heuristic(t).data(chosen(p,2),1);
if (demand.heuristic(t).data(i,8) ~=...
 demand.heuristic(t).data(i,9))...
&& (demand.heuristic(t).data(chosen(p,1),8) ~=...
demand.heuristic(t).data(chosen(p,2),8))...
&& (demand.heuristic(t).data(i,9) ~=...
 demand.heuristic(t).data(chosen(p,1),8))
    if (maxpax1 <= aircraft.types(t,4)) &&...
     (maxpax2 <= aircraft.types(t,4))
        flag = timetest(t,[i;chosen(p,1);chosen(p,2)]);
        if (flag == 1)
        selected(t,[i;chosen(p,1);chosen(p,2)]);
        numsel = numsel + 1;
        else
            numrel = numrel + 1;
```

```
        end;
    end;
end;
            end;
        end;
    end;
end;
display('** Dual-linking Completed.');
disp(strcat('   Number of Links: ',num2str(numsel)));
disp(strcat('   Links removed due to timing: ',num2str(numrel)));
end

function selected(f,selection)
%SELECTED Updates all CIP parameter databases with
% selected flight route
% (Only works for linking 3 flights into a sequence of 2)

global demand;
global cipdata;
global flights;
costn = 0;
timen = 0;
varnum = size(cipdata.cost,1)+1;
[timen,costn] = timecost(f,demand.heuristic(f).data(selection(1,1),8),...
demand.heuristic(f).data(selection(1,1),9),timen,costn);
[timen,costn] = timecost(f,demand.heuristic(f).data(selection(3,1),8),...
demand.heuristic(f).data(selection(3,1),9),timen,costn);
cipdata.selected(varnum,1) = f;
ki = [];
for j=1:1:size(selection,1)
    ki = [ki demand.heuristic(f).data(selection(j,1),7)];
    y = 10;
    foundblank = 0;
    if (demand.heuristic(f).data(selection(j,1),y) ~= 0)
        while (foundblank == 0)
            if (demand.heuristic(f).data(selection(j,1),y) == 0)
foundblank = 1;
            else
ki = [ki demand.heuristic(f).data(selection(j,1),y)];
y = y + 1;
            end;
        end;
    end;
```

```
end;
for r=1:1:size(ki,2)
    flights(ki(1,r),1) = flights(ki(1,r),1) + 1;
    foundblank = 0;
    y = 2;
    while (foundblank == 0)
        if (flights(ki(1,r),y) == 0)
            flights(ki(1,r),y) = varnum;
            foundblank = 1;
        else
            if (y == size(flights,2))
foundblank = 2;
            else
y = y + 1;
            end;
        end;
    end;
    if (foundblank == 2)
        flights(ki(1,r),(size(flights,2)+1)) = varnum;
    end;
    cipdata.selected(varnum,r+1) = ki(1,r);
end;
cipdata.time(varnum,f) = timen;
cipdata.cost(varnum,1) = costn;
end

function out = timetest(f,selection)
%TIMETEST Tests whether linking is possible with respect to timing
%constraints
global demand;
global aircraft;
global distances;
flag = 0;
sizesel = size(selection,1);
locations = zeros(sizesel,2);
for i=1:1:sizesel
locations(i,1) = demand.heuristic(f).data(selection(i,1),8);
locations(i,2) = demand.heuristic(f).data(selection(i,1),9);
end;
if (locations(1,1) == locations(2,1))
    if (locations(1,2) == locations(3,1))
        starter = 1;
        mid = 2;
```

```
        ender = 3;
    else
        mid = 1;
        starter = 2;
        ender = 3;
    end;
else
    if (locations(1,1) == locations(3,1))
        if (locations(1,2) == locations(2,1))
            starter = 1;
            ender = 2;
            mid = 3;
        else
            starter = 3;
            ender = 2;
            mid = 1;
        end;
    else
        if (locations(1,1) == locations(2,2))
            starter = 2;
            ender = 1;
            mid = 3;
        else
            starter = 3;
            ender = 1;
            mid = 2;
        end;
    end;
end;
flight1times = zeros(1,4);
flight1times(1,1) = max(demand.heuristic(f).data(selection(starter,1),2),...
demand.heuristic(f).data(selection(mid,1),2));
flight1times(1,2) = min(demand.heuristic(f).data(selection(starter,1),3),...
demand.heuristic(f).data(selection(mid,1),3));
flight1times(1,3) = max(demand.heuristic(f).data(selection(starter,1),4),...
demand.heuristic(f).data(selection(ender,1),2));
flight1times(1,4) = min(demand.heuristic(f).data(selection(starter,1),5),...
demand.heuristic(f).data(selection(ender,1),3));
flight2times(1,1) = max(demand.heuristic(f).data(selection(mid,1),4),...
demand.heuristic(f).data(selection(ender,1),4));
flight2times(1,2) = min(demand.heuristic(f).data(selection(mid,1),5),...
demand.heuristic(f).data(selection(ender,1),5));
allowance = 0; %% time allowance for leeway in time constraints (minutes)
```

```
if (flight1times(1,1) <= timeadd(allowance,flight1times(1,2)))...
        && (flight1times(1,3) <= timeadd(allowance,flight1times(1,4)))...
        && (flight2times(1,1) <= timeadd(allowance,flight2times(1,2)))
    time1mins = (distances(demand.heuristic(f).data...
    (selection(starter,1),8),demand.heuristic(f).data...
    (selection(starter,1),9))/aircraft.types(f,1))*60;
    time2mins = (distances(demand.heuristic(f).data...
    (selection(ender,1),8),demand.heuristic(f).data...
    (selection(ender,1),9))/aircraft.types(f,1))*60;
    if (timeadd(allowance,timeadd(time1mins,flight1times(1,2))) >=...
         flight1times(1,3))...
            && (timeadd(time1mins,flight1times(1,1)) <=...
             timeadd(allowance,flight1times(1,4)))...
            && (timeadd(allowance,timeadd(time2mins,flight1times(1,4)))...
             >=flight2times(1,1))...
            && (timeadd(time2mins,flight1times(1,3)) <=...
             timeadd(allowance,flight2times(1,2)))
        flag = 1;
    end;
end;
if (flag == 1)
    out = 1;
else
    out = 0;
end;
end

function chainer(linkstart)
%CHAINER Joins links & singles together to form chains
global cipdata;
global demand;
global aircraft;

chsnum = 0;
tempvarlocs = zeros(size(cipdata.selected,1)-linkstart,100);
for i=linkstart:1:size(cipdata.selected,1)
    tempvarlocs(i-linkstart+1,1) = i; %varnum
    tempvarlocs(i-linkstart+1,2) = cipdata.selected(i,1); %plane
    locs = [demand.sorted(cipdata.selected(i,2),8)...
     demand.sorted(cipdata.selected(i,2),9)];
    for j=3:1:size(cipdata.selected,2)
        if (cipdata.selected(i,j) ~= 0)
            if (demand.sorted(cipdata.selected(i,j),9) ~= locs(1,end))
```

```
locs(1,end+1) = demand.sorted(cipdata.selected(i,j),9);
            end;
        end;
    end;
    tempvarlocs(i-linkstart+1,3:size(locs,2)+2) = locs(1,:);
end;

for i=1:1:size(tempvarlocs,1)
    before = [];
    after = [];
    for j=1:1:size(tempvarlocs,1)
        if (i ~= j)
            if (tempvarlocs(i,2) == tempvarlocs(j,2))
if (tempvarlocs(i,3) == tempvarlocs(j,3))...
        && (tempvarlocs(i,4) == tempvarlocs(j,5))
    paxst = [];
    paxen = [];
    for ki=2:1:size(cipdata.selected,2)
        if (cipdata.selected(tempvarlocs(i,1),ki) > 0)
            if (demand.sorted(cipdata.selected(tempvarlocs(i,1),ki),8)...
             == tempvarlocs(j,3))
paxst(1,end+1) = cipdata.selected(tempvarlocs(i,1),ki);
            end;
        end;
        if (cipdata.selected(tempvarlocs(j,1),ki) > 0)
            if (demand.sorted(cipdata.selected(tempvarlocs(j,1),ki),8)...
             == tempvarlocs(j,3))
paxst(1,end+1) = cipdata.selected(tempvarlocs(j,1),ki);
            end;
        end;
        if (cipdata.selected(tempvarlocs(i,1),ki) > 0)
            if (demand.sorted(cipdata.selected(tempvarlocs(i,1),ki),9)...
             == tempvarlocs(j,5))...
    || (demand.sorted(cipdata.selected(tempvarlocs(i,1),ki),9)...
     == tempvarlocs(i,5))
if (demand.sorted(cipdata.selected(tempvarlocs(i,1),ki),8)...
 == tempvarlocs(j,3))...
        || (demand.sorted(cipdata.selected(tempvarlocs(i,1),ki),8)...
         == tempvarlocs(j,4))
    paxen(1,end+1) = cipdata.selected(tempvarlocs(i,1),ki);
end;
            end;
        end;
```

```
        if (cipdata.selected(tempvarlocs(j,1),ki) > 0)
            if (demand.sorted(cipdata.selected(tempvarlocs(j,1),ki),9)...
             == tempvarlocs(j,5))...
    || (demand.sorted(cipdata.selected(tempvarlocs(j,1),ki),9)...
     == tempvarlocs(i,5))
if (demand.sorted(cipdata.selected(tempvarlocs(j,1),ki),8)...
 == tempvarlocs(j,3))...
        || (demand.sorted(cipdata.selected(tempvarlocs(j,1),ki),8)...
         == tempvarlocs(j,4))
    paxen(1,end+1) = cipdata.selected(tempvarlocs(j,1),ki);
end;
            end;
        end;
    end;
    if (size(paxen,2) > 0) && (size(paxst,2) > 0)
        for pi=2:1:size(cipdata.selected,2)
            for pj=1:1:size(paxst,2)
if (cipdata.selected(i,pi) == paxst(1,pj))
    paxst(1,pj) = 0;
end;
            end;
        end;
        paxst2 = [];
        if (paxst(1,1) == 0)
            paxst = paxst(1,2:end);
        end;
        paxst2(1,end+1) = paxst(1,1);
        for jk=2:1:size(paxst,2)
            found = 0;
            for jc=1:1:size(paxst2,2)
if (paxst(1,jk) == paxst2(1,jc))
    found = 1;
end;
            end;
            if (found == 0) && (paxst(1,jk) ~= 0)
paxst2(1,end+1) = paxst(1,jk);
            end;
        end;
        paxen2 = [];
        paxen2(1,end+1) = paxen(1,1);
        for jk=2:1:size(paxen,2)
            found = 0;
            for jc=1:1:size(paxen2,2)
```

```
if (paxen(1,jk) == paxen2(1,jc))
    found = 1;
end;
            end;
            if (found == 0)
paxen2(1,end+1) = paxen(1,jk);
            end;
        end;
        pax1 = 0;
        for jk=1:1:size(paxst2,2)
            pax1 = pax1 + demand.sorted(paxst2(1,jk),1);
        end;
        pax2 = 0;
        for jk=1:1:size(paxen2,2)
            pax2 = pax2 + demand.sorted(paxen2(1,jk),1);
        end;
        if (pax1 <= aircraft.types(tempvarlocs(i,2),4)) &&...
         (pax2 <= aircraft.types(tempvarlocs(i,2),4))
            before(1,end+1) = tempvarlocs(j,1);
        end;
    end;
else
    if (tempvarlocs(i,4) == tempvarlocs(j,3))...
            && (tempvarlocs(i,5) == tempvarlocs(j,5))
        %%after
        paxst = [];
        paxen = [];
        for ki=2:1:size(cipdata.selected,2)
            if (cipdata.selected(tempvarlocs(i,1),ki) > 0)
if (demand.sorted(cipdata.selected(tempvarlocs(i,1),ki),8) ==...
 tempvarlocs(j,3))
    paxst(1,end+1) = cipdata.selected(tempvarlocs(i,1),ki);
end;
            end;
            if (cipdata.selected(tempvarlocs(j,1),ki) > 0)
if (demand.sorted(cipdata.selected(tempvarlocs(j,1),ki),8) ==...
 tempvarlocs(j,3))
    paxst(1,end+1) = cipdata.selected(tempvarlocs(j,1),ki);
end;
            end;
            if (cipdata.selected(tempvarlocs(i,1),ki) > 0)
if (demand.sorted(cipdata.selected(tempvarlocs(i,1),ki),9) ==...
 tempvarlocs(j,5))
```

```
    paxen(1,end+1) = cipdata.selected(tempvarlocs(i,1),ki);
end;
            end;
            if (cipdata.selected(tempvarlocs(j,1),ki) > 0)
if (demand.sorted(cipdata.selected(tempvarlocs(j,1),ki),9) ==...
 tempvarlocs(j,5))
    paxen(1,end+1) = cipdata.selected(tempvarlocs(j,1),ki);
end;
            end;
        end;
            if (size(paxen,2) > 0) && (size(paxst,2) > 0)
paxst2 = [];
paxst2(1,end+1) = paxst(1,1);
for jk=2:1:size(paxst,2)
    found = 0;
    for jc=1:1:size(paxst2,2)
        if (paxst(1,jk) == paxst2(1,jc))
            found = 1;
        end;
    end;
    if (found == 0)
        paxst2(1,end+1) = paxst(1,jk);
    end;
end;
for pi=2:1:size(cipdata.selected,2)
    for pj=1:1:size(paxen,2)
        if (cipdata.selected(i,pi) == paxen(1,pj))
            paxen(1,pj) = 0;
        end;
    end;
end;
paxen2 = [];
if (paxen(1,1) == 0)
    paxen = paxen(1,2:end);
end;
paxen2(1,end+1) = paxen(1,1);
for jk=2:1:size(paxen,2)
    found = 0;
    for jc=1:1:size(paxen2,2)
        if (paxen(1,jk) == paxen2(1,jc))
            found = 1;
        end;
    end;
```

```
    if (found == 0) && (paxen(1,jk) ~= 0)
        paxen2(1,end+1) = paxen(1,jk);
    end;
end;
pax1 = 0;
for jk=1:1:size(paxst2,2)
    pax1 = pax1 + demand.sorted(paxst2(1,jk),1);
end;
pax2 = 0;
for jk=1:1:size(paxen2,2)
    pax2 = pax2 + demand.sorted(paxen2(1,jk),1);
end;
if (pax1 <= aircraft.types(tempvarlocs(i,2),4)) && (pax2 <=...
 aircraft.types(tempvarlocs(i,2),4))
    after(1,end+1) = tempvarlocs(j,1);
end;
            end;
    end;
end;
            end;
        end;
    end;
    %%make all possible linkchains
    if (size(before,2) ~= 0)
        for k=1:1:size(before,2)
            pos = 0;
            for po=1:1:size(tempvarlocs,1)
if (tempvarlocs(po,1) == before(1,k))
    pos = po;
end;
            end;
            if (pos ~= 0)
times = [600 1800 600 1800;600 1800 600 1800];
for kp=2:1:size(cipdata.selected,2)
    if (cipdata.selected(before(1,k),kp) ~= 0)
        if (demand.sorted(cipdata.selected(before(1,k),kp),8) ==...
         tempvarlocs(pos,3))
            if (demand.sorted(cipdata.selected(before(1,k),kp),2)...
             > times(1,1))
times(1,1) = demand.sorted(cipdata.selected(before(1,k),kp),2);
            end;
            if (demand.sorted(cipdata.selected(before(1,k),kp),3)...
             < times(1,2))
```

```
times(1,2) = demand.sorted(cipdata.selected(before(1,k),kp),3);
             end;
        end;
        if (demand.sorted(cipdata.selected(before(1,k),kp),9)...
         == tempvarlocs(pos,5))
            if (demand.sorted(cipdata.selected(before(1,k),kp),4)...
             > times(1,3))
times(1,3) = demand.sorted(cipdata.selected(before(1,k),kp),4);
            end;
            if (demand.sorted(cipdata.selected(before(1,k),kp),5)...
             < times(1,4))
times(1,4) = demand.sorted(cipdata.selected(before(1,k),kp),5);
            end;
        end;
    end;
    %%
    if (cipdata.selected(tempvarlocs(i,1),kp) ~= 0)
        if (demand.sorted(cipdata.selected(tempvarlocs(i,1),kp),8)...
         == tempvarlocs(i,3))
            if (demand.sorted(cipdata.selected(tempvarlocs(i,1),kp),2)...
             > times(2,1))
times(2,1) = demand.sorted(cipdata.selected(tempvarlocs(i,1),kp),2);
            end;
            if (demand.sorted(cipdata.selected(tempvarlocs(i,1),kp),3)...
             < times(2,2))
times(2,2) = demand.sorted(cipdata.selected(tempvarlocs(i,1),kp),3);
            end;
        end;
        if (demand.sorted(cipdata.selected(tempvarlocs(i,1),kp),9)...
         == tempvarlocs(i,4))
            if (demand.sorted(cipdata.selected(tempvarlocs(i,1),kp),4)...
             > times(2,3))
times(2,3) = demand.sorted(cipdata.selected(tempvarlocs(i,1),kp),4);
            end;
            if (demand.sorted(cipdata.selected(tempvarlocs(i,1),kp),5)...
             < times(2,4))
times(2,4) = demand.sorted(cipdata.selected(tempvarlocs(i,1),kp),5);
            end;
        end;
    end;
end;
if ((times(1,1) <= times(2,1)) && (times(1,2) >= times(2,1)))...
 || ((times(1,1) >= times(2,1)) && (times(1,1) <= times(2,2)))
```

```
    if ((times(1,3) <= times(2,3)) && (times(1,4) >= times(2,3)))...
     || ((times(1,3) >= times(2,3)) && (times(1,3) <= times(2,4)))
        %% can add in link.
        chsnum = chsnum + 1;
        selectorchains(tempvarlocs(i,1),before(1,k),1,[tempvarlocs(pos,3)...
         tempvarlocs(pos,4) tempvarlocs(i,4) tempvarlocs(i,5)]);
    end;
end;
            end;
        end;
    end;
    if (size(after,2) ~= 0)
        for k=1:1:size(after,2)
            pos = 0;
            for po=1:1:size(tempvarlocs,1)
if (tempvarlocs(po,1) == after(1,k))
    pos = po;
end;
            end;
            if (pos ~= 0)
times = [600 1800 600 1800;600 1800 600 1800];
for kp=2:1:size(cipdata.selected,2)
    if (cipdata.selected(after(1,k),kp) ~= 0)
        if (demand.sorted(cipdata.selected(after(1,k),kp),8)...
         == tempvarlocs(pos,3))
            if (demand.sorted(cipdata.selected(after(1,k),kp),2)...
             > times(1,1))
times(1,1) = demand.sorted(cipdata.selected(after(1,k),kp),2);
            end;
            if (demand.sorted(cipdata.selected(after(1,k),kp),3)...
             < times(1,2))
times(1,2) = demand.sorted(cipdata.selected(after(1,k),kp),3);
            end;
        end;
        if (demand.sorted(cipdata.selected(after(1,k),kp),9)...
         == tempvarlocs(pos,5))
            if (demand.sorted(cipdata.selected(after(1,k),kp),4)...
             > times(1,3))
times(1,3) = demand.sorted(cipdata.selected(after(1,k),kp),4);
            end;
            if (demand.sorted(cipdata.selected(after(1,k),kp),5)...
             < times(1,4))
times(1,4) = demand.sorted(cipdata.selected(after(1,k),kp),5);
```

```
            end;
        end;
    end;
    %%
    if (cipdata.selected(tempvarlocs(i,1),kp) ~= 0)
        if (demand.sorted(cipdata.selected(tempvarlocs(i,1),kp),8)...
         == tempvarlocs(i,4))
            if (demand.sorted(cipdata.selected(tempvarlocs(i,1),kp),2)...
             > times(2,1))
times(2,1) = demand.sorted(cipdata.selected(tempvarlocs(i,1),kp),2);
            end;
            if (demand.sorted(cipdata.selected(tempvarlocs(i,1),kp),3)...
             < times(2,2))
times(2,2) = demand.sorted(cipdata.selected(tempvarlocs(i,1),kp),3);
            end;
        end;
        if (demand.sorted(cipdata.selected(tempvarlocs(i,1),kp),9)...
         == tempvarlocs(i,5))
            if (demand.sorted(cipdata.selected(tempvarlocs(i,1),kp),4)...
             > times(2,3))
times(2,3) = demand.sorted(cipdata.selected(tempvarlocs(i,1),kp),4);
            end;
            if (demand.sorted(cipdata.selected(tempvarlocs(i,1),kp),5)...
             < times(2,4))
times(2,4) = demand.sorted(cipdata.selected(tempvarlocs(i,1),kp),5);
            end;
        end;
    end;
end;
if ((times(1,1) <= times(2,1)) && (times(1,2) >= times(2,1)))...
 || ((times(1,1) >= times(2,1)) && (times(1,1) <= times(2,2)))
    if ((times(1,3) <= times(2,3)) && (times(1,4) >= times(2,3)))...
     || ((times(1,3) >= times(2,3)) && (times(1,3) <= times(2,4)))
        %% can add in link.
        chsnum = chsnum + 1;
        selectorchains(tempvarlocs(i,1),after(1,k),2,[tempvarlocs(i,3)...
         tempvarlocs(i,4) tempvarlocs(pos,4) tempvarlocs(pos,5)]);
    end;
end;
            end;
        end;
    end;
    if (size(before,2) ~= 0) && (size(after,2) ~= 0)
```

```
        %
        for k=1:1:size(before,2)
            for jk=1:1:size(after,2)
posb = 0;
posa = 0;
for po=1:1:size(tempvarlocs,1)
    if (tempvarlocs(po,1) == before(1,k))
        posb = po;
    end;
    if (tempvarlocs(po,1) == after(1,jk))
        posa = po;
    end;
end;
if (posa ~= 0) && (posb ~= 0)
    times = [600 1800 600 1800;600 1800 600 1800];
    for kp=2:1:size(cipdata.selected,2)
        if (cipdata.selected(after(1,jk),kp) ~= 0)
            if (demand.sorted(cipdata.selected(after(1,jk),kp),8)...
             == tempvarlocs(posa,3))
if (demand.sorted(cipdata.selected(after(1,jk),kp),2)...
 > times(1,1))
times(1,1) = demand.sorted(cipdata.selected(after(1,jk),kp),2);
end;
if (demand.sorted(cipdata.selected(after(1,jk),kp),3)..
 < times(1,2))
times(1,2) = demand.sorted(cipdata.selected(after(1,jk),kp),3);
end;
            end;
            if (demand.sorted(cipdata.selected(after(1,jk),kp),9)...
             == tempvarlocs(posa,5))
if (demand.sorted(cipdata.selected(after(1,jk),kp),4) > times(1,3))
times(1,3) = demand.sorted(cipdata.selected(after(1,jk),kp),4);
end;
if (demand.sorted(cipdata.selected(after(1,jk),kp),5) < times(1,4))
    times(1,4) = demand.sorted(cipdata.selected(after(1,jk),kp),5);
end;
            end;
        end;
        %%
        if (cipdata.selected(tempvarlocs(i,1),kp) ~= 0)
            if (demand.sorted(cipdata.selected(tempvarlocs(i,1),kp),8)...
             == tempvarlocs(i,4))
if (demand.sorted(cipdata.selected(tempvarlocs(i,1),kp),2) > times(2,1))
```

```
    times(2,1) = demand.sorted(cipdata.selected(tempvarlocs(i,1),kp),2);
end;
if (demand.sorted(cipdata.selected(tempvarlocs(i,1),kp),3) < times(2,2))
    times(2,2) = demand.sorted(cipdata.selected(tempvarlocs(i,1),kp),3);
end;
            end;
            if (demand.sorted(cipdata.selected(tempvarlocs(i,1),kp),9)...
             == tempvarlocs(i,5))
if (demand.sorted(cipdata.selected(tempvarlocs(i,1),kp),4) > times(2,3))
    times(2,3) = demand.sorted(cipdata.selected(tempvarlocs(i,1),kp),4);
end;
if (demand.sorted(cipdata.selected(tempvarlocs(i,1),kp),5) < times(2,4))
    times(2,4) = demand.sorted(cipdata.selected(tempvarlocs(i,1),kp),5);
end;
            end;
        end;
    end;
    if ((times(1,1) <= times(2,1)) && (times(1,2) >= times(2,1)))...
     || ((times(1,1) >= times(2,1)) && (times(1,1) <= times(2,2)))
        if ((times(1,3) <= times(2,3)) && (times(1,4) >= times(2,3)))...
         || ((times(1,3) >= times(2,3)) && (times(1,3) <= times(2,4)))

            times = [600 1800 600 1800;600 1800 600 1800];
            for kp=2:1:size(cipdata.selected,2)
if (cipdata.selected(before(1,k),kp) ~= 0)
    if (demand.sorted(cipdata.selected(before(1,k),kp),8)...
     == tempvarlocs(posb,3))
        if (demand.sorted(cipdata.selected(before(1,k),kp),2)...
         > times(1,1))
            times(1,1) = demand.sorted(cipdata.selected...
            (before(1,k),kp),2);
        end;
        if (demand.sorted(cipdata.selected(before(1,k),kp),3)...
         < times(1,2))
            times(1,2) = demand.sorted(cipdata.selected...
            (before(1,k),kp),3);
        end;
    end;
    if (demand.sorted(cipdata.selected(before(1,k),kp),9)...
     == tempvarlocs(posb,5))
        if (demand.sorted(cipdata.selected(before(1,k),kp),4) > times(1,3))
            times(1,3) = demand.sorted(cipdata.selected(before(1,k),kp),4);
        end;
```

```
        if (demand.sorted(cipdata.selected(before(1,k),kp),5) < times(1,4))
            times(1,4) = demand.sorted(cipdata.selected(before(1,k),kp),5);
        end;
    end;
end;
%%
if (cipdata.selected(tempvarlocs(i,1),kp) ~= 0)
    if (demand.sorted(cipdata.selected(tempvarlocs(i,1),kp),8)...
     == tempvarlocs(i,3))
        if (demand.sorted(cipdata.selected(tempvarlocs(i,1),kp),2)...
         > times(2,1))
            times(2,1) = demand.sorted(cipdata.selected(tempvarlocs(i,1),kp),2);
        end;
        if (demand.sorted(cipdata.selected(tempvarlocs(i,1),kp),3)...
         < times(2,2))
            times(2,2) = demand.sorted(cipdata.selected...
            (tempvarlocs(i,1),kp),3);
        end;
    end;
    if (demand.sorted(cipdata.selected(tempvarlocs(i,1),kp),9)...
     == tempvarlocs(i,4))
        if (demand.sorted(cipdata.selected(tempvarlocs(i,1),kp),4)...
         > times(2,3))
            times(2,3) = demand.sorted(cipdata.selected...
            (tempvarlocs(i,1),kp),4);
        end;
        if (demand.sorted(cipdata.selected(tempvarlocs(i,1),kp),5)...
         < times(2,4))
            times(2,4) = demand.sorted(cipdata.selected...
            (tempvarlocs(i,1),kp),5);
        end;
    end;
end;
            end;
            if ((times(1,1) <= times(2,1)) && (times(1,2)...
             >= times(2,1)))...
             || ((times(1,1) >= times(2,1)) && (times(1,1)...
              <= times(2,2)))
if ((times(1,3) <= times(2,3)) && (times(1,4) >= times(2,3)))...
 || ((times(1,3) >= times(2,3)) && (times(1,3) <= times(2,4)))
    %% can add in double link.
    chsnum = chsnum + 1;
    selectorchains(tempvarlocs(i,1),[before(1,k) after(1,jk)],3,...
```

```
    [tempvarlocs(posb,3) tempvarlocs(posb,4) tempvarlocs(posb,5)...
     tempvarlocs(posa,4) tempvarlocs(posa,5)]);
end;
            end;
        end;
    end;
end;
            end;
        end;
    end;
    %%
end;

cipdata.linklocs = tempvarlocs;
display('** Chaining of Links and Flights Completed.');
disp(strcat('   Chains created:',num2str(chsnum)));
end

function selectorchains(main,inserts,type,sequence)
%SELECTORCHAINS Summary of this function goes here
%   Detailed explanation goes here
global cipdata;
global flights;
global demand;
varnum = size(cipdata.selected,1) + 1;
timen = 0;
costn = 0;
pl = cipdata.selected(main,1);
if (type == 1)
    [timen,costn] = timecost(pl,sequence(1,1),sequence(1,2),timen,costn);
    [timen,costn] = timecost(pl,sequence(1,2),sequence(1,3),timen,costn);
    [timen,costn] = timecost(pl,sequence(1,3),sequence(1,4),timen,costn);
    cipdata.time(varnum,pl) = timen;
    cipdata.cost(varnum,1) = costn;
    flts = [];
    for rp=2:1:size(cipdata.selected,2)
        if (cipdata.selected(main,rp) ~= 0)
            flts(1,end+1) = cipdata.selected(main,rp);
        end;
        if (cipdata.selected(inserts(1,1),rp) ~= 0)
            flts(1,end+1) = cipdata.selected(inserts,rp);
        end;
    end;
```

```
    flts2(1,1) = flts(1,1);
    for ki=2:1:size(flts,2)
        found = 0;
        for kj=1:1:size(flts2,2)
            if (flts2(1,kj) == flts(1,ki))
found = 1;
            end;
        end;
        if (found == 0)
            flts2(1,end+1) = flts(1,ki);
        end;
    end;
    sels = [];
    for st=1:1:size(sequence,2)
        for en=1:1:size(sequence,2)
            if (st < en)
for i=1:1:size(flts2,2)
    if (demand.sorted(flts2(1,i),8) == sequence(1,st)) &&...
     (demand.sorted(flts2(1,i),9) == sequence(1,en))
        sels(1,end+1) = flts2(1,i);
    end;
end;
            end;
        end;
    end;
    sels2(1,1) = sels(1,1);
    ki = 2;
    while (ki ~= size(sels,2)+1)
        kj = 1;
        while (kj ~= size(sels2,2)+1)
            if (sels2(1,kj) == sels(1,ki))
sels2 = [sels2(1,1:kj-1) sels2(1,kj+1:end)];
            end;
            kj = kj + 1;
        end;
        sels2(1,end+1) = sels(1,ki);
        ki = ki + 1;
    end;
    cipdata.selected(varnum,1) = pl;
    cipdata.selected(varnum,2:size(sels2,2)+1) = sels2(1,:);
    for i=1:1:size(sels2,2)
        flights(sels2(1,i),1) = flights(sels2(1,i),1) + 1;
        flights(sels2(1,i),flights(sels2(1,i),1)+1) = varnum;
```

```
    end;
elseif (type == 2)
    [timen,costn] = timecost(pl,sequence(1,1),sequence(1,2),timen,costn);
    [timen,costn] = timecost(pl,sequence(1,2),sequence(1,3),timen,costn);
    [timen,costn] = timecost(pl,sequence(1,3),sequence(1,4),timen,costn);
    cipdata.time(varnum,pl) = timen;
    cipdata.cost(varnum,1) = costn;
    flts = [];
    for rp=2:1:size(cipdata.selected,2)
        if (cipdata.selected(main,rp) ~= 0)
            flts(1,end+1) = cipdata.selected(main,rp);
        end;
        if (cipdata.selected(inserts(1,1),rp) ~= 0)
            flts(1,end+1) = cipdata.selected(inserts,rp);
        end;
    end;
    flts2(1,1) = flts(1,1);
    for ki=2:1:size(flts,2)
        found = 0;
        for kj=1:1:size(flts2,2)
            if (flts2(1,kj) == flts(1,ki))
found = 1;
            end;
        end;
        if (found == 0)
            flts2(1,end+1) = flts(1,ki);
        end;
    end;
    sels = [];
    for st=1:1:size(sequence,2)
        for en=1:1:size(sequence,2)
            if (st < en)
for i=1:1:size(flts2,2)
    if (demand.sorted(flts2(1,i),8) == sequence(1,st)) &&...
     (demand.sorted(flts2(1,i),9) == sequence(1,en))
        sels(1,end+1) = flts2(1,i);
    end;
end;
            end;
        end;
    end;
    sels2(1,1) = sels(1,1);
    ki = 2;
```

```
    while (ki ~= size(sels,2)+1)
        kj = 1;
        while (kj ~= size(sels2,2)+1)
            if (sels2(1,kj) == sels(1,ki))
sels2 = [sels2(1,1:kj-1) sels2(1,kj+1:end)];
            end;
            kj = kj + 1;
        end;
        sels2(1,end+1) = sels(1,ki);
        ki = ki + 1;
    end;
    cipdata.selected(varnum,1) = pl;
    cipdata.selected(varnum,2:size(sels2,2)+1) = sels2(1,:);
    for i=1:1:size(sels2,2)
        flights(sels2(1,i),1) = flights(sels2(1,i),1) + 1;
        flights(sels2(1,i),flights(sels(1,i),1)+1) = varnum;
    end;
elseif (type == 3)
    [timen,costn] = timecost(pl,sequence(1,1),sequence(1,2),timen,costn);
    [timen,costn] = timecost(pl,sequence(1,2),sequence(1,3),timen,costn);
    [timen,costn] = timecost(pl,sequence(1,3),sequence(1,4),timen,costn);
    [timen,costn] = timecost(pl,sequence(1,4),sequence(1,5),timen,costn);
    cipdata.time(varnum,pl) = timen;
    cipdata.cost(varnum,1) = costn;

    flts = [];
    for rp=2:1:size(cipdata.selected,2)
        if (cipdata.selected(main,rp) ~= 0)
            flts(1,end+1) = cipdata.selected(main,rp);
        end;
        if (cipdata.selected(inserts(1,1),rp) ~= 0)
            flts(1,end+1) = cipdata.selected(inserts(1,1),rp);
        end;
        if (cipdata.selected(inserts(1,2),rp) ~= 0)
            flts(1,end+1) = cipdata.selected(inserts(1,2),rp);
        end;
    end;
    flts2(1,1) = flts(1,1);
    for ki=2:1:size(flts,2)
        found = 0;
        for kj=1:1:size(flts2,2)
            if (flts2(1,kj) == flts(1,ki))
found = 1;
```

```
            end;
        end;
        if (found == 0)
            flts2(1,end+1) = flts(1,ki);
        end;
    end;
    sels = [];
    for st=1:1:size(sequence,2)
        for en=1:1:size(sequence,2)
            if (st < en)
for i=1:1:size(flts2,2)
    if (demand.sorted(flts2(1,i),8) == sequence(1,st)) &&...
     (demand.sorted(flts2(1,i),9) == sequence(1,en))
        sels(1,end+1) = flts2(1,i);
    end;
end;
            end;
        end;
    end;
    sels2(1,1) = sels(1,1);
    ki = 2;
    while (ki ~= size(sels,2)+1)
        kj = 1;
        while (kj ~= size(sels2,2)+1)
            if (sels2(1,kj) == sels(1,ki))
sels2 = [sels2(1,1:kj-1) sels2(1,kj+1:end)];
            end;
            kj = kj + 1;
        end;
        sels2(1,end+1) = sels(1,ki);
        ki = ki + 1;
    end;
    cipdata.selected(varnum,1) = pl;
    cipdata.selected(varnum,2:size(sels2,2)+1) = sels2(1,:);
    for i=1:1:size(sels2,2)
        flights(sels2(1,i),1) = flights(sels2(1,i),1) + 1;
        flights(sels2(1,i),flights(sels2(1,i),1)+1) = varnum;
    end;
end;
end

function [timen,costn] = timecost(f,st,en,timen,costn)
%TIMECOST Calculates current time and cost for a flight leg and adds it to
```

```
%the previously calculated time and cost for that grouping.
global distances;
global aircraft;
timec = distances(st,en)/aircraft.types(f,1);
timen = timec*60 + timen + aircraft.types(f,8) + aircraft.types(f,9);
cost = timec*(aircraft.types(f,5) + aircraft.types(f,6))...
 + aircraft.types(f,10);
costn = costn + cost;
end

function Tnodes
global newcipdata;
global aircraft;
global locations;
count = 0;
for t = 1 : size(aircraft.types,1)
    newcipdata.air(t).times = zeros(size(locations.data,1),73);
for i = 1 : 73
    for j = 1 : size(locations.data,1)
        count = count + 1;
        newcipdata.air(t).times(j,i) = count;
    end
end
end

function newtimes
%NEWTIMES Time constraints using the 10 minute block principle
global cipdata;
global demand;
global aircraft;
global locations;
global newcipdata;
display('** Creating CIP Data.');
display('** Creating Ten Minute Time Blocks.');
varnum = 0;
ddata = sortrows(demand.data,7);
%%preallocate newcipdata!!!
newcipdata.varstats = zeros(0,5);
newcipdata.cost = zeros(0,1);
newcipdata.selected = zeros(0,size(cipdata.selected,2));
newcipdata.time = sparse(0,size(aircraft.types,1));

for t = 1 : size(aircraft.types,1)
```

```
    newcipdata.takeoff = zeros(73*size(locations.data,1)...
    *size(aircraft.types,1)+size(aircraft.types,1),1200);
    newcipdata.landing = zeros(73*size(locations.data,1)...
    *size(aircraft.types,1)+size(aircraft.types,1),1200);
end

%%
for i=1:1:size(cipdata.time,1) %1
    for j=1:1:size(cipdata.time,2) %2
        if (cipdata.time(i,j) ~= 0) %3
            finished = 0;
            locs = zeros(15,1);
            locs(1,1) = ddata(cipdata.selected(i,2),8);
            k = 2;
            k2 = 1;
            sm = size(cipdata.selected,2);
            while (finished ~= 1)
if (k ~= sm + 1)
    if (cipdata.selected(i,k) ~= 0)
        if (ddata(cipdata.selected(i,k),9) ~= locs(end,1))
            k2 = k2 + 1;
            locs(k2,1) = ddata(cipdata.selected(i,k),9);
        end;
        k = k + 1;
    else
        finished = 1;
    end;
else
    finished = 1;
end;
            end;
            locs = matshrink(locs);
            timecons = [600 1800 600 1800]; % unconstrained times
            kend = 0;
            for qw=2:1:size(cipdata.selected,2)
if (kend == 0)
    if (cipdata.selected(i,qw) ~= 0)
    if (demand.sorted(cipdata.selected(i,qw),8) ~= locs(1,1))
        kend = qw;
    end;
    else
        kend = qw;
    end;
```

```
end;
            end;
            kstr = 0;
            for qw=size(cipdata.selected,2):-1:2
if (kstr == 0)
    if (cipdata.selected(i,qw) ~= 0)
     if (demand.sorted(cipdata.selected(i,qw),9) ~= locs(end,1))
        kstr = qw;
    end;
    end;
end;
            end;
            if (kstr == 0)
kstr = 2;
            end;
            for p=2:1:k-1
if (ddata(cipdata.selected(i,p),8) == locs(1,1))
    if (p <= kend)
    if (ddata(cipdata.selected(i,p),2) > timecons(1))
        timecons(1) = ddata(cipdata.selected(i,p),2);
    end;
    if (ddata(cipdata.selected(i,p),3) < timecons(2))
        timecons(2) = ddata(cipdata.selected(i,p),3);
    end;
    end
else
               % if (ddata(cipdata.selected(i,p),9) ==...
                locs(end,1))
    if (p >= kstr)
    if (ddata(cipdata.selected(i,p),9) == locs(end,1))
        if (ddata(cipdata.selected(i,p),4) > timecons(3))
            timecons(3) = ddata(cipdata.selected(i,p),4);
        end;
        if (ddata(cipdata.selected(i,p),5) < timecons(4))
            timecons(4) = ddata(cipdata.selected(i,p),5);
        end;
    end;
    end;
end
            end;
            for h=1:1:4
t1 = timecons(h);
hr = floor(t1/100);
```

```
mins = t1 - hr*100;
t2 = (hr-6)*60 + mins;
timecons(h) = t2;
            end;
            t1 = ceil(timecons(1)/10);
            t2 = floor(timecons(2)/10);
            startloc = locs(1,1);
            endloc = locs(end,1);
            t = j;
            for tp=t1:1:t2 %%2 for half size
if (tp*10 + cipdata.time(i,j) >= timecons(3))...
        && (tp*10 + cipdata.time(i,j) <= timecons(4))
    added = 1;
    varnum = varnum + 1;
    endtime = ceil(cipdata.time(i,j)/10)+tp;
    if (endtime > timecons(4))
        endtime = timecons(4);
    end;
    newcipdata.cost(varnum,1) = cipdata.cost(i,1);
    newcipdata.time(varnum,j) = cipdata.time(i,j);
    newcipdata.selected(varnum,:) = cipdata.selected(i,:);
    %%
    newcipdata.varstats(varnum,1) = j;
    newcipdata.varstats(varnum,2) = locs(1,1);%%start
    newcipdata.varstats(varnum,3) = locs(end,1);%%end
    newcipdata.varstats(varnum,4) = tp;%%start-time
    newcipdata.varstats(varnum,5) = endtime;%%end-time

    takepos = newcipdata.air(t).times(startloc,tp + 1);
    newcipdata.takeoff(takepos,1) = newcipdata.takeoff(takepos,1) + 1;
    newcipdata.takeoff(takepos,newcipdata.takeoff(takepos,1)+1) = varnum;

    landpos = newcipdata.air(t).times(endloc,endtime+1);
    newcipdata.landing(landpos,1) = newcipdata.landing(landpos,1) + 1;
    newcipdata.landing(landpos,newcipdata.landing(landpos,1)+1) = varnum;

end;
            end;
        end; %3
    end; %2
end; %1
%%
```

```
newcipdata.startconnect = zeros(1,2);
selrow = size(newcipdata.selected,1);
newcipdata.startconnect(1,1) = selrow + 1;
varnum = selrow;
for t = 1 : size(aircraft.types,1)
    for i = 1 : size(aircraft.heuristic(t).data,1)
        varnum = varnum + 1;
        newcipdata.cost(varnum,1) = 0;
        newcipdata.time(varnum,t) = 0;
        newcipdata.selected(varnum,1) = t;

        newcipdata.varstats(varnum,1) = t;
        newcipdata.varstats(varnum,2) = aircraft.heuristic(t).data(i,3);
        newcipdata.varstats(varnum,3) = aircraft.heuristic(t).data(i,3);
        newcipdata.varstats(varnum,4) = 0;%%start-time
        newcipdata.varstats(varnum,5) = 0;%%end-time

        landpos = newcipdata.air(t).times(aircraft.heuristic(t).data(i,3),1);
        newcipdata.landing(landpos,1) = newcipdata.landing(landpos,1) + 1;
        newcipdata.landing(landpos,newcipdata.landing(landpos,1)+1) = varnum;
    end
end
 newcipdata.startconnect(1,2) = size(newcipdata.selected,1);

for t = 1 : size(aircraft.types,1)
    newcipdata.landing = matshrinkright(newcipdata.landing);
    newcipdata.takeoff = matshrinkright(newcipdata.takeoff);
end
newcipdata.varstats = matshrink(newcipdata.varstats);
newcipdata.selected = matshrink(newcipdata.selected);
newcipdata.cost = matshrink(newcipdata.cost);

end

function [out] = matshrinkright(in)
%MATSHRINKRIGHT Summary of this function goes here
%   Detailed explanation goes here
%%part of matshrink reused below
[n,m] = size(in);
foundm = 0;
zero = zeros(n,m);
for i=1:1:m
    if (foundm == 0)
```

```
       if (in(:,i) == zero(:,1))
           if (i ~= m)
           if (in(:,i+1) == zero(:,1))
               if (i ~= m-1)
   if (in(:,i+2) == zero(:,1))
       foundm = i;
   end;
               else
               foundm=i;
               end;
           end;
           else
               foundm=i;
           end;
       end;
    end
end;
if (foundm == 0)
    foundm = m+1;
end;
out = in(:,1:foundm-1);
end

function [outmat] = matshrink(inmat)
%MATSHRINK Removes outer zero rows and columns from a matrix
%   Useful for when a matrix is preallocated for a potentially large set
%   of data and a large unused section of the matrix is taking up excess
%   memory on the computer.
[n,m] = size(inmat);
foundn = 0;
foundm = 0;
zero = zeros(n,m);
for i=1:1:n
    if (foundn == 0)
       if (inmat(i,:) == zero(1,:))
           if (i ~= n)
           if (inmat(i+1,:) == zero(1,:))
               if (i ~= n-1)
   if (inmat(i+2,:) == zero(1,:))
       foundn = i;
   end;
               else
               foundn=i;
```

```
                end;
            end;
            else
                foundn=i;
            end;
        end;
    end
end;
if (foundn == 0)
    foundn = n+1;
end;
for i=1:1:m
    if (foundm == 0)
       if (inmat(:,i) == zero(:,1))
           if (i ~= m)
           if (inmat(:,i+1) == zero(:,1))
               if (i ~= m-1)
   if (inmat(:,i+2) == zero(:,1))
       foundm = i;
   end;
               else
               foundm=i;
               end;
           end;
           else
               foundm=i;
           end;
       end;
    end
end;
if (foundm == 0)
    foundm = m+1;
end;
outmat = inmat(1:foundn-1,1:foundm-1);
end

function Tsource
global locations;
global distances;
global aircraft;
global newcipdata;
global demand;
display('** Creating Empty Flights.');
```

```
varnum = size(newcipdata.selected,1);
newcipdata.startsource = zeros(1,2);
newcipdata.startsource(1,1) = size(newcipdata.selected,1) + 1;
szdem = size(demand.sorted,1);
newcipdata.sourceflight = zeros(0,9);

for t = 1 : size(aircraft.types,1)
    for i = 1 : size(locations.data,1)
        startloc = i;
        for j = 1 : size(locations.data,1)
            if i == j
continue;
            end
            endloc = j;
            time1 = distances(i,j)/aircraft.types(t,1);
            time2 = time1*60 + aircraft.types(t,8) + aircraft.types(t,9);
            cost = time1*(aircraft.types(t,5) + aircraft.types(t,6)) +...
             aircraft.types(t,10);
            for tp = 0 : 1 : 72
if (tp*10 + time2 < 720)
    varnum = varnum + 1;
    endtime = ceil(time2/10)+tp;
    sourcenum = size(newcipdata.sourceflight,1) + 1 + szdem;

    newcipdata.cost(varnum,1) = cost;
    newcipdata.time(varnum,t) = time2;
    newcipdata.selected(varnum,1) = t;
    newcipdata.selected(varnum,2) = sourcenum;

    newcipdata.varstats(varnum,1) = t;
    newcipdata.varstats(varnum,2) = startloc;%%startloc
    newcipdata.varstats(varnum,3) = endloc;%%endloc
    newcipdata.varstats(varnum,4) = tp;%%start-time
    newcipdata.varstats(varnum,5) = endtime;%%end-time

    takepos = newcipdata.air(t).times(startloc,tp + 1);
    newcipdata.takeoff(takepos,1) = newcipdata.takeoff(takepos,1) + 1;
    newcipdata.takeoff(takepos,newcipdata.takeoff(takepos,1)+1) = varnum;


    landpos = newcipdata.air(t).times(endloc,endtime+1);
    newcipdata.landing(landpos,1) = newcipdata.landing(landpos,1) + 1;
```

```
    newcipdata.landing(landpos,newcipdata.landing(landpos,1)+1) = varnum;
end
            end
        end
    end
end
newcipdata.startsource(1,2) = size(newcipdata.selected,1);

air = newcipdata.startconnect(1,1) : 1 : newcipdata.startconnect(1,2);


for t = 1 : size(aircraft.types,1)
    num = 0;
    for a = 1 : size(air,2)
    if newcipdata.selected(air(1,a),1) == t
      num = newcipdata.takeoff(73*size(locations.data,1)...
      *size(aircraft.types,1)+t,1);
      newcipdata.takeoff(73*size(locations.data,1)...
      *size(aircraft.types,1)+t,1)...
       = newcipdata.takeoff(73*size(locations.data,1)...
       *size(aircraft.types,1)+t,1) + 1;
      newcipdata.takeoff(73*size(locations.data,1)...
      *size(aircraft.types,1)+t,num+2)...
       = air(1,a);
    end
    end
end

function Tgroundnodes
global newcipdata;
global aircraft;
global locations;


newcipdata.takeoffground = zeros(size(locations.data,1)...
*size(aircraft.types,1),73);
newcipdata.landingground = zeros(size(locations.data,1)...
*size(aircraft.types,1),73);

varnum = 0;

for t = 1 : size(aircraft.types,1)
    for i = 1 : 73
```

```
    for j = size(locations.data,1)*(t-1) +1 : size(locations.data,1)*t
        varnum = varnum + 1;
        newcipdata.takeoffground(j,i) = varnum;
        if(i~=73)
        newcipdata.landingground(j,i+1) = varnum;
        end;
    end
    end
end

newcipdata.groundsink = zeros(1,size(locations.data,1)...
*size(aircraft.types,1));
newcipdata.groundsink(1,:) = newcipdata.takeoffground(:,73);

function Trecordconnector
global newcipdata;
global demand;

newcipdata.flights = zeros(size(demand.sorted,1),60000);
szselc = size(newcipdata.selected,2);
%szdem = size(demand.sorted,1);
display('** Record all the Information of the Variables Created.');

%recording booking variables
for i = 1 : size(demand.sorted,1)
    pos = zeros(1,szselc);
    pos = find(sum(newcipdata.selected(:,2:end) == i,2)>0);
    pos = matshrink(pos);
    newcipdata.flights(i,1) = length(pos);
    newcipdata.flights(i,2:length(pos)+1) = pos';
end

newcipdata.flights = matshrink(newcipdata.flights);
display('** Recorded All the Variables for The CIP Solver');

function TSefzplcreate
%ZPLCREATE Creates the zimpl files needed

global aircraft;
global newcipdata;
global locations;

display('** Generating ZIMPL Files.');
```

```
%% Create Single Files
zpl = fopen('files/output/Tcharter.txt','w+');
cos = fopen('files/output/costs.dat','w+');
rot = fopen('files/output/routes.txt','w+');
rod = fopen('files/output/routes.dat','w+');
fle = fopen('files/output/fleets.dat','w+');
tkt = fopen('files/output/takeoffs.txt','w+');
tkd = fopen('files/output/takeoffs.dat','w+');
ldt = fopen('files/output/landings.txt','w+');
ldd = fopen('files/output/landings.dat','w+');
tgt = fopen('files/output/takeoffsgnd.txt','w+');
tgd = fopen('files/output/takeoffsgnd.dat','w+');
lgt = fopen('files/output/landingsgnd.txt','w+');
lgd = fopen('files/output/landingsgnd.dat','w+');



%% Modify Flights (remove blank entries)

tempfl = newcipdata.flights;
i = 1;
ended = 0;
while (ended == 0)
    if (tempfl(i,1) == 0)
        tempfl(i,:) = [];
    end;
    if (i == size(tempfl,1))
        ended = 1;
    end;
    i = i + 1;
end;



%% Create ZPL file.
fprintf(zpl,'%s\n','# Charter aircraft scheduling assignment problem');
fprintf(zpl,'%s\n','# Tanya La Foy (0309425X)');
fprintf(zpl,'%s\n','# University of the Witwatersrand');
fprintf(zpl,'%s\n','');
fprintf(zpl,'%s\n','set V  := { read "costs.dat" as "<1n>" comment "#"};...
# of variables');
fprintf(zpl,'%s','set VG := { 1 to');
```

```
fprintf(zpl,' %s ',num2str(73*size(locations.data,1)*size(aircraft.types,1)...
+size(aircraft.types,1)));
fprintf(zpl,'%s\n','};# of ground variables');
fprintf(zpl,'%s\n','set R  := { read "routes.dat" as "<1n>" comment "#"};...
# of routes');
fprintf(zpl,'%s','set RW := { 1 to');
fprintf(zpl,' %s ',num2str((size(tempfl,2)-1)));
fprintf(zpl,'%s\n','};# width of routes parameter');
fprintf(zpl,'%s\n','set A  := { read "fleets.dat" as "<1n>" comment "#"};...
# of fleet');
fprintf(zpl,'%s','set ND := { 1 to');
fprintf(zpl,' %s ',num2str(size(locations.data,1)*73*size(aircraft.types,1) +...
 size(aircraft.types,1)));
fprintf(zpl,'%s\n','};# width of nodes');
fprintf(zpl,'%s','set ND2 := { 1 to');
fprintf(zpl,' %s ',num2str(size(locations.data,1)*73*size(aircraft.types,1)));
fprintf(zpl,'%s\n','};# width of nodes');
fprintf(zpl,'%s','set TKW := { 1 to');
fprintf(zpl,' %s ',num2str(size(locations.data,1)*73*size(aircraft.types,1)));
fprintf(zpl,'%s\n','};# width of takeoffs parameter');
fprintf(zpl,'%s','set LDW := { 1 to');
fprintf(zpl,' %s ',num2str(size(locations.data,1)*73*size(aircraft.types,1)));
fprintf(zpl,'%s\n','};# width of landings parameter');
fprintf(zpl,'%s','set TGW := { 1 to');
fprintf(zpl,' %s ',num2str(1));
fprintf(zpl,'%s\n','};# width of takeoffs ground parameter');
fprintf(zpl,'%s','set LGW := { 1 to');
fprintf(zpl,' %s ',num2str(1));
fprintf(zpl,'%s\n','};# width of landings ground parameter');

for t = 1 : size(aircraft.types,1)
   fprintf(zpl,'%s','set GSINK');
   fprintf(zpl,'%s', strcat(num2str(t)));
   fprintf(zpl,'%s',':= ');
   fprintf(zpl,'%s',strcat('{',num2str(newcipdata.air(t).times(1,73)),' to'));
   fprintf(zpl,' %s\n ',strcat(num2str(newcipdata.air(t).times...
   (size(locations.data,1),73)),'};# ground arcs to the super sink node'));
end

fprintf(zpl,'%s\n','');
fprintf(zpl,'%s\n','param c[V]      := read "costs.dat" as "<1n> 2n"...
 comment "#"; # cost for each variable');
fprintf(zpl,'%s\n','param rnum[R]   := read "routes.dat" as "<1n> 2n"...
```

```
 comment "#"; # of flights per route');
fprintf(zpl,'%s\n','param air[A]    := read "fleets.dat" as "<1n> 2n"...
 comment "#"; # of aircraft per fleet type');
fprintf(zpl,'%s\n','param tkn[ND]   := read "takeoffs.dat" as "<1n> 2n"...
 comment "#"; # of flights taking-off at a specific node');
fprintf(zpl,'%s\n','param lnd[ND]   := read "landings.dat" as "<1n> 2n"...
 comment "#"; # of flights landing at a specific node');
fprintf(zpl,'%s\n','param tgn[ND]   := read "takeoffsgnd.dat" as "<1n> 2n"...
 comment "#"; # of flights takeoff at a specific ground node');
fprintf(zpl,'%s\n','param lgn[ND]   := read "landingsgnd.dat" as "<1n> 2n"...
 comment "#"; # of flights landing at a specific ground node');

fprintf(zpl,'%s\n','');
fprintf(zpl,'%s\n','include "routes.txt"; # Parameter table containing the...
 variable number for each route.');
fprintf(zpl,'%s\n','include "takeoffs.txt"; ');
fprintf(zpl,'%s\n','include "landings.txt"; ');
fprintf(zpl,'%s\n','include "takeoffsgnd.txt"; ');
fprintf(zpl,'%s\n','include "landingsgnd.txt"; ');

fprintf(zpl,'%s\n','');
fprintf(zpl,'%s\n','var x[V]  binary;');
fprintf(zpl,'%s\n','var y[VG]  integer;');
fprintf(zpl,'%s\n','');
fprintf(zpl,'%s\n','minimize  cost: sum <i> in V : c[i] * x[i];');

fprintf(zpl,'%s\n','subto routes: forall <i> in R do');
fprintf(zpl,'%s\n','    sum <j> in { 1 to rnum[i]} :...
x[routes[i,j]] == 1;');

fprintf(zpl,'%s\n','subto flow: forall <i> in ND2 do');
fprintf(zpl,'%s\n','    sum <j> in { 1 to tkn[i]} :...
x[takeoffs[i,j]]+sum <k> in { 1 to tgn[i]} : y[takeoffsground[i,k]]...
 -  sum <w> in { 1 to lnd[i]} : x[landings[i,w]] -   sum <q> ...
 in { 1 to lgn[i]} : y[landingsground[i,q]] == 0;');

for t = 1 : size(aircraft.types,1)
    fprintf(zpl,'%s\n','');
    fprintf(zpl,'%s','subto flow_end');
    fprintf(zpl,'%s', strcat(num2str(t)));
    fprintf(zpl,'%s\n',strcat(': y[',num2str(73*size(locations.data,1)...
    *size(aircraft.types,1) + t),'] - sum <g> in GSINK',num2str(t),':...
           y[g] == 0;'));
```

```
    %fprintf(zpl,'%s', strcat(num2str(t)));
    %fprintf(zpl,'%s',strcat(': y[g] == 0;'));
end

for t = 1 : size(aircraft.types,1)
    fprintf(zpl,'%s\n','');
    fprintf(zpl,'%s','subto flow_start');
    fprintf(zpl,'%s', strcat(num2str(t)));
    fprintf(zpl,'%s\n',strcat(': y[',num2str(73*size(locations.data,1)...
    *size(aircraft.types,1) + t),'] - sum <j> in {1 to tkn[',...
    num2str(73*size(locations.data,1)*size(aircraft.types,1) + t),']}...
     :x[takeoffs[',num2str(73*size(locations.data,1)...
     *size(aircraft.types,1) + t),',j]] == 0;'));
end

for t = 1 : size(aircraft.types,1)
    fprintf(zpl,'%s\n','');
    fprintf(zpl,'%s','subto avail');
    fprintf(zpl,'%s', strcat(num2str(t)));
    fprintf(zpl,'%s\n',strcat(': y[',num2str(73*size(locations.data,1)...
    *size(aircraft.types,1) + t),'] <= air[',num2str(t),'];'));
end


fprintf(zpl,'%s\n','subto aircraft: ');
for i = newcipdata.startconnect(1,1) : newcipdata.startconnect(1,2)-1
   fprintf(zpl,'%s',strcat(' x[',num2str(i),'] +'));
end
i = newcipdata.startconnect(1,2);
fprintf(zpl,'%s\n',strcat(' x[',num2str(i),'] <= ',...
num2str(size(aircraft.data,1)),';'));

fprintf(zpl,'%s','# End of File');
fclose(zpl);
display('   Created Tcharter.zpl file.');

%% Create costs.dat file
fprintf(cos,'%s\n','#Variable Costs');
varcount = size(newcipdata.cost,1);
for i=1:1:varcount
    fprintf(cos,'%s ',num2str(i));
    if (i ~= varcount)
        fprintf(cos,'%s\n',num2str(round(newcipdata.cost(i,1))));
```

```
    else
        fprintf(cos,'%s',num2str(round(newcipdata.cost(i,1))));
    end;
end;
fclose(cos);
display('   Created costs.dat file.');

%% Create routes.txt file
fprintf(rot,'%s\n','param routes[R*RW] := ');
szcon1 = size(newcipdata.flights,1);
szcon2 = size(newcipdata.flights,2);
fprintf(rot,'%s',strcat('<',num2str(1),',',num2str(1),'> ',...
num2str(newcipdata.flights(1,2))));
for j=1:1:szcon1
    for k=2:1:szcon2
        if (j~=1) || (k ~= 2)
            if (newcipdata.flights(j,k) ~= 0)
            fprintf(rot,', %s',strcat('<',num2str(j),',',num2str(k-1),...
            '> ',num2str(newcipdata.flights(j,k))));
            end;
        end;
    end;
end;
fprintf(rot,'%s',';');
fclose(rot);
display('   Created routes.txt file.');

%% Create routes.dat file
fprintf(rod,'%s\n','# Number of Flights Per Route');
siflights1 = size(newcipdata.flights,1);
siflights2 = size(newcipdata.flights,2);
for i=1:1:siflights1
    if (newcipdata.flights(i,1) ~= 0)
        if (i ~= siflights1)
            fprintf(rod,'%s ',num2str(i));
            fprintf(rod,'%s\n',num2str(newcipdata.flights(i,1)));
        else
            fprintf(rod,'%s ',num2str(i));
            fprintf(rod,'%s',num2str(newcipdata.flights(i,1)));
        end;
    end;
end;
fclose(rod);
```

```
display('   Created routes.dat file.');

%% Create fleets.dat file
fprintf(fle,'%s\n','#Number of Aircrat in Per Fleet Type');
varcount = size(aircraft.types,1);
for i=1:1:varcount
    fprintf(fle,'%s ',num2str(i));
    if (i ~= varcount)
        fprintf(fle,'%s\n',num2str(aircraft.types(i,11)));
    else
        fprintf(fle,'%s',num2str(aircraft.types(i,11)));
    end;
end;
fclose(fle);
display('   Created fleet.dat file.');

%% Create takeoffs.txt file
fprintf(tkt,'%s\n','param takeoffs[ND*TKW] := ');
szcon1 = size(newcipdata.takeoff,1);
szcon2 = size(newcipdata.takeoff,2);
fprintf(tkt,'%s',strcat('<',num2str(1),',',num2str(1),'> ',...
num2str(newcipdata.takeoff(1,2))));
for j=1:1:szcon1
    for k=2:1:szcon2
        if (j~=1) || (k ~= 2)
            %if (newcipdata.takeoff(j,k) ~= 0)
            fprintf(tkt,', %s',strcat('<',num2str(j),',',...
            num2str(k-1),'> ',num2str(newcipdata.takeoff(j,k))));
            %end;
        end;
    end;
end;
fprintf(tkt,'%s',';');
fclose(tkt);
display('   Created takeoffs.txt file.');

%% Create takeoffs.dat file
fprintf(tkd,'%s\n','# Number Takeoffs of Flights Per Node');
siflights1 = size(newcipdata.takeoff,1);
siflights2 = size(newcipdata.takeoff,2);
for i=1:1:siflights1
        if (i ~= siflights1)
            fprintf(tkd,'%s ',num2str(i));
```

```
            fprintf(tkd,'%s\n',num2str(newcipdata.takeoff(i,1)));
        else
            fprintf(tkd,'%s ',num2str(i));
            fprintf(tkd,'%s',num2str(newcipdata.takeoff(i,1)));
        end;
end;
fclose(tkd);
display('   Created takeoffs.dat file.');

%% Create landings.txt file
fprintf(ldt,'%s\n','param landings[ND*LDW] := ');
szcon1 = size(newcipdata.landing,1);
szcon2 = size(newcipdata.landing,2);
fprintf(ldt,'%s',strcat('<',num2str(1),',',num2str(1),'> ',...
num2str(newcipdata.landing(1,2))));
for j=1:1:szcon1
    for k=2:1:szcon2
        if (j~=1) || (k ~= 2)
            %if (newcipdata.landing(j,k) ~= 0)
            fprintf(ldt,', %s',strcat('<',num2str(j),',',...
            num2str(k-1),'> ',num2str(newcipdata.landing(j,k))));
            %end
        end;
    end;
end;
fprintf(ldt,'%s',';');
fclose(ldt);
display('   Created landings.txt file.');

%% Create landings.dat file
fprintf(ldd,'%s\n','# Number of Landings Flights Per Node');
siflights1 = size(newcipdata.landing,1);
siflights2 = size(newcipdata.landing,2);
for i=1:1:siflights1
        if (i ~= siflights1)
            fprintf(ldd,'%s ',num2str(i));
            fprintf(ldd,'%s\n',num2str(newcipdata.landing(i,1)));
        else
            fprintf(ldd,'%s ',num2str(i));
            fprintf(ldd,'%s',num2str(newcipdata.landing(i,1)));
        end;
end;
fclose(ldd);
```

```
display('   Created landings.dat file.');

%% Create takeoffsgnd.txt file
fprintf(tgt,'%s\n','param takeoffsground[ND*TGW] := ');
count = 0;
count = count + 1;
fprintf(tgt,'%s',strcat('<',num2str(1),',',num2str(1),'> '...
,num2str(count)));
for j=2:1:73*size(locations.data,1)*size(aircraft.types,1)
    count = count + 1;
    fprintf(tgt,', %s',strcat('<',num2str(j),',',num2str(1),...
    '> ',num2str(count)));
end;
fprintf(tgt,'%s',';');
fclose(tgt);
display('   Created takeoffsgnds.txt file.');

%% Create takeoffsgnd.dat file
fprintf(tgd,'%s\n','# Number Ground of Takeoff per Node');

for i=1:1:73*size(locations.data,1)*size(aircraft.types,1)
            fprintf(tgd,'%s ',num2str(i));
            fprintf(tgd,'%s\n',num2str(1));
end;
fclose(tgd);
display('   Created takeoffsgnd.dat file.');


%% Create landingsgnd.txt file
fprintf(lgt,'%s\n','param landingsground[ND*LGW] := ');
count = 0;
first=0;
for t = 1 : size(aircraft.types,1)
for i = 1:1:73
    for j= (size(locations.data,1)*(t-1) + 1) : 1 : (size(locations.data,1)*t)
        count = count + 1;
        if(newcipdata.landingground(j,i)~=0)
            if(first==0)
               fprintf(lgt,'%s',strcat('<',num2str(count),',',num2str(1),...
               '> ',num2str(newcipdata.landingground(j,i))));
               first=1;
           else
            fprintf(lgt,', %s',strcat('<',num2str(count),',',num2str(1),...
```

```
            '> ',num2str(newcipdata.landingground(j,i))));
           end
        end;
    end;
end;
end
fprintf(lgt,'%s',';');
fclose(lgt);
display('   Created landingsgnds.txt file.');


%% Create landingsgnd.dat file
fprintf(lgd,'%s\n','# Number Ground of Landing per Node');
for t = 1 : size(aircraft.types,1)
for i= newcipdata.air(t).times(1,1) : 1 :...
 newcipdata.air(t).times(size(locations.data,1),1)
            fprintf(lgd,'%s ',num2str(i));
            fprintf(lgd,'%s\n',num2str(0));
end;
for i = newcipdata.air(t).times(1,2) : 1 :...
 newcipdata.air(t).times(size(locations.data,1),73)
            fprintf(lgd,'%s ',num2str(i));
            fprintf(lgd,'%s\n',num2str(1));
end;
end

fclose(lgd);
display('   Created landingsgnd.dat file.');
```

Bibliography

[1] Shangyao Yan, Chich-Hwang Tseng. A passenger demand model for airline flight scheduling and fleet routing. Computers and Operations Research 2002;29:1559-1581.

[2] Nikolaos Papadakos. Integrated airline scheduling: Decomposition and acceleration techniques. Research Report, IC-PARC, Imperial College, UK, 2006.

[3] D. Espinoza, R. Garcia, M. Goycoolea, G.L. Nemhauser, M.W.P. Savelsbergh. Per-seat, on-demand air transportation part I: problem and an integer multicommodity flow model. Transportation Science 2008;42:263-278.

[4] Dayjet Corporation. www.dayjet.com

[5] Teodorovic D, Krcmar-Nozic E. Multicriteria model to determine flight frequencies on an airline network under competitive conditions. Transportation Science 1989;23:14-25.

[6] Hane C, Barnhart C, Johnson EL, Marsten R, Nemhauser GL, Sigismondi G. The fleet assignment probelm: solving a large-scale integer program Mathematical Programming Study 1995;70:211-32.

[7] Keskinocak P and Tayur S. Scheduling of Time-Share Aircraft. Transportation Science 1998;3:277-294.

[8] Ronen D. Scheduling Charter Aircraft. Journal of Operational Research Society 2000;51:258-262.

[9] Martin C, Jones D and Keskinocak P. Bitwise Fractional Airline Optimizer, INFORMS, San Jose November 17-20,2002.

[10] Martin C, Jones D and Keskinocak P. Optimizing On-Demand aircraft Schedules for Fractional Aircraft Operators, Interfaces 2003;33:22-35.

[11] Hicks R, Madrid R, Milligan C, Pruneau R, Kanaley M, Dumas Y, Lacroix C, Desrosiers J and Soumis F. Bombardier Flexjet Significantly Improves Its Fractional Aircraft Ownership Operations, Interfaces 2005;35(1):49-60.

[12] MPS file format. http://lpsolve.sourceforge.net/5.5/mps-format.htm.

[13] Koch T. ZIMPL User Guide. 18 September 2006.

[14] Levin A. Scheduling and fleet routing models for transportation systems. Transporation Science 1971;5(3), 232-255.

[15] Sherali H, Bish E and Zhu X. Airline scheduling concepts, models, and alogorithms. European Journal of Operations Research 2006;172(1):1 - 30.

[16] Marsten R and Shepardson F. Exact Solution of Crew Scheduling Problems Using the Set Partitioning Model: Recent Successful Applications, Research Report, Graduate School of Business, Stanford University, 1979.

[17] Schaefer A, Johnson E, Kleywegt A and Nemhauser G. Airline crew scheduling under uncertainty. Transportation Science 2005;39(3):340-348.

[18] Burke E, Patrick, Kleywegt A., and Nemhauser G.L. A multi-objective approach for robust airline scheduling. Computers and Operations Research. 2010;37(5):822-832.

[19] Lohatepanont M and Barnhart C. Airline scheduling planning integrated models and algorithms for schedule design and fleet assignment. Transporation Science. 2004;38:19-32.

[20] Barnhart C, Lu F and Shenoi R. Integrated airline schedule planning. Operations Research in the airline industry. 1998;9:384-403.

[21] Klabjan D. Large scale models in the airline industry, column generation, kluwer academic publishers. Available at https://netfiles.uiuc.edu/klabjan/www.

[22] Nemhauser G.L. and Wolsey L.A 1999. Integer and combinatorial optimization. John Wiley and Sons Inc.

[23] Rosenberger J, Johnson E and Nemhauser G. Routing aircraft for airline recovery. Transportation Science. Nov 2003;37(4):408-421.

[24] Rabetanety A. Airline Schedule Planning Integrated Flight Schedule Design and Product Line Design. University Karlsruhe (TH), PhD thesis, 2006.

[25] Flexjet. http://www.flexjet.com..

[26] Sefofane. http://Sefofane.com..

[27] Mou D and Zhane Z. The integrated model of airline fleet assignment and aircraft routing based on flight cycle. 2010 International Conference on Management Science & Engineering. Nov 2010, 24-26.

[28] Berge M. Timetable Optimization: Formualtion, Solution Approaches, and Computational Issues. AGIFORS proceeding. 1994;341-357.

[29] Lan S. Planning for Robust Airline Operations: Optimizing Aircraft Routing and FLight Departure Times to Acheive Minimum Passenger Disruptions. Massachusetts Institute of Technology. Jun 2003.

[30] Rexing B, Barnhart C, Kniker T, Jarrah A and Krishnamurthy N. Airline Fleet Assignment with Time Windows. Tranportation Science. 2000;34:1-20.

[31] Marsten R, Subramanian and Gibbons L. Junior Analyst Extraordinare (Jane). AGIFORS PRO., Athens. 1996;247-259.

[32] Subramanian R, Scheff R. Jr., Quillinan J, Wiper D and Marsten R. Coldstart: Fleet Assignment at Delta Airlines. Interfaces 1994;24(1):104-120.

[33] Flow network. http://en.wikipedia.org/wiki/Flow_network.