DISRUPTION MANAGEMENT IN THE AIRLINE INDUSTRY

Michael Løve &
Kim Riis Sørensen

LYNGBY 2001
MASTER THESIS
NR. 02/2001

IMM

Printed by IMM, DTU

Preface

This M.Sc. thesis has been prepared by Michael Løve and Kim Riis Sørensen during the period from the 1^{st} of August, 2000 to the 28^{th} of February, 2001. The work has been carried out at the Department of Informatics and Mathematical Modelling (IMM) at the Technical University of Denmark (DTU).

We have been supervised by our main advisor professor Jens Clausen and assistant research professor Jesper Larsen. This thesis is the final requirement to obtain the degree: Master of Science in Engineering.

Readers of this thesis are assumed to have some knowledge of Operations Research (OR). However, numerous terms are not assumed to be known because they are adapted from the airline industry. These terms are all defined in the back of this thesis and shown in **boldface** the first time they appear.

Michael Løve & Kim Riis Sørensen
Lyngby, February 28, 2001

Acknowledgements

We would first of all like to thank Jens Clausen and Jesper Larsen. Throughout these past 7 months they have commented, criticized and encouraged our work in a motivating and serious manner. Thank you.

We also owe a large thanks to Thomas Stidsen, Brian Kallehauge and Jesper Hansen for their efforts in helping us to grasp the computer systems at IMM. Also thanks to Allan Larsen, Alex Ross and Nicki Davis – all of whom supplied us with airline industry insights.

A large thanks also goes to Patricia Løve for proof-reading the entire thesis – and even picking out little mathematical inconsistencies.

We would also like to thank everybody including Rasmus Bøg, Mette Krogh-Jespersen, Niels Jørgensen and ORSCG for making it thoroughly enjoyable to work at IMM. We would also like to dispense one piece of advise to ORSCG: Never relent. In time, you too may master the art.

And finally, we would like to thank Hanne Riis Sørensen and Ursula Lindhart for taking extra care of us and filling in where we did not.

Michael Løve & Kim Riis Sørensen
Lyngby, February 28, 2001

Abstract

Disruption management in the airline industry involves decisions concerning crew and aircraft assignments in situations where unforeseen events have disrupted the existing flight schedules, e.g. bad weather causing flight delays. Disruption management aims to recover these flight schedules through a series of reassignments of crew and aircraft.

The first part of this thesis concentrates on disruption management where only aircraft assignments are changed – the Dedicated Aircraft Recovery Problem. A heuristic is implemented which can generate revised flight schedules of a good quality in less than 5 seconds on average when applied to 25 different flight schedules with disruptions.

In the second part of the thesis, an outline of an extended heuristic is given. Here disruption management includes the possibility of both aircraft *and* crew reassignments - the Integrated Crew and Aircraft Recovery Problem. The outline presents a detailed description of how to construct such a heuristic.

Throughout all the work presented here, the focus has been to develop methods which are simple, flexible and fast.

Keywords: Disruption Management, Optimization, Airline Planning, Dedicated Aircraft Recovery, Integrated Recovery, Heuristic.

Contents

Chapter 1

Introduction

The ability to plan effectively is a major factor for gaining competitive advantage in the airline industry today, and the use of operations research is quickly becoming one of the most important tools to do this.

The product that airlines produce is – simply put – passenger and cargo transportation. Within the industry itself there is fierce competition, but this is aggravated by the fact that other modes of transportation are increasingly able to compete with aircraft. Especially trains are quickly becoming a serious competitor, largely due to the emergence of high speed rail networks. Buses too, are serious competitors because of the low costs involved. Lastly, cars are a competitor on shorter distances.

Customers choosing a mode of transportation and a specific service provider will typically focus on the following parameters:

- Punctuality
- Functionality
- Quality
- Price

For an airline this means that there are a series of factors, that combined constitute a good product. Among other things, they must be on-time, their schedules must suit the needs of the customers, check-in and baggage handling must be effective, the level of service on board has to be good, booking a flight must be easy and finally, the cost to the customers must be low. Naturally there are trade-offs between these factors and no airline can excel in all of them.

Airlines require a large number of resources to deliver a product that includes all of the factors just described. The most obvious ones are aircraft, crew, and airport facilities such as runways and gates. However, a wide variety of other resources are also needed including training facilities, catering services, maintenance facilities, etc.

A competitive edge in the airline industry lies in how well they are able to utilize all these resources. Particularly aircraft and airport facilities are very expensive to operate, so resources must be tightly coupled. It is here the complexity of operating an airline arises along with the need for planning. The complexity arises because of several factors:

- An endless number of rules, regulations, union demands and preferences apply to crew and aircraft schedules. The complexity of simply producing cohesive flight schedules that respect all these restrictions is a daunting task.
- Airlines operate across time zones, cultures and continents – the operational scope is enormous.
- While considering the above two factors, the airlines must create very tight plans that utilize resources to the furthest extent possible. This results in plans with very little slack.
- Airlines operate in an unpredictable environment where **disruptions** often occur. The near absence of slack in the flight schedules often cause small disruptions to have multiple knock-on effects further down in the flight schedule. These have to be repaired while respecting all the factors just described.

It is evident that planning in the airline industry takes place on many levels. So far, operations research has mainly been used in planning with a longer time horizon. Some of these planning horizons are illustrated in figure 1.1.

Construction of timetables, **aircraft rosters**, and **crew rosters** are some of the tasks in airlines where operations research is used today. It is obvious that these planning activities take place before flights are carried out.

This thesis focuses solely on the planning issues that arise in the immediate vicinity of the departure time. In short, each departure is a product of years of careful planning. Therefore, when disruptions occur, flight controllers want to return to the original schedule as quickly as possible – and that is exactly what the methods developed in this thesis focus on.

Figure 1.1: *Airline planning horizons.*

So far only limited research has been carried out in this field. However, there appears to be an increased demand for decision support systems that can assist flight controllers when they try to recover flight schedules with disruptions.

Chapter 2

Problem Description

The purpose of this chapter is to give a precise definition of the problems dealt with in this thesis. This includes a definition of the terms and concepts used as well as a description of the context in which the problems exists.

2.1 The Dedicated Aircraft Recovery Problem

The Dedicated Aircraft Recovery Problem (DARP) is not a problem, which is well defined and widely recognized. Because of this, there are several names for principally the same problem: DARP is also known as the Flight Operations Decision Problem (FODP), the Operational Daily Airline Scheduling Problem (ODASP), the Daily Aircraft Routing and Scheduling Problem (DARSP), Airline Operations Control Center Problem (AOCCP), Airline Schedule Perturbation Problem (ASPP), etc.. However, all these names basically refer to the same problem. Throughout this thesis, the term DARP will therefore be used to describe the problem, which is formally defined in figure 2.1. The definition given here contains several concepts that are also important to define:

Flight Schedule: For a given airline, the flight schedule includes all flights between any two destinations, the original departure and arrival times, the expected flight durations and the **tail assignments**.

Given an original flight schedule and one or more disruptions, the Dedicated Aircraft Recovery Problem consists of:

- delaying flights
- cancelling flights
- swapping **aircraft to flight assignments**

in order to create a more preferable revised flight schedule.

Figure 2.1: *Definition of DARP.*

Disruption: Any situation where one or more activities in one or more of the key resource areas (e.g. crew or aircraft) have deviated from the resource plan. Subsequent activities in the affected lines of work either cannot start on time – or can start on time, but only after controller intervention.

Delaying a flight: Purposefully preventing a flight from departing on time constitutes delaying the flight.

Cancelling a flight: If no aircraft is assigned to a given flight, this constitutes a cancellation.

Swapping: The concept of swapping is central to Dedicated Aircraft Recovery (DAR). Although the concept is very simple, it is important to define it as in figure 2.2. Here, aircraft 1 and 2 are situated at airport 1 and they are assigned to flight 1 and 2 respectively. Flights are static throughout this thesis, i.e. they always fly to the originally scheduled destination. Conversely, aircraft do not always undertake the originally intended flights. In figure 2.2 the flights undertaken by aircraft 1 and 2 are swapped so that aircraft 1 ends at destination B and vice versa. Performing any number of swaps is also known as *rotating* aircraft.

2.1.1 Decision Costs

Another central element in DARP are the decision costs. In operations research costs must reflect how *preferable* each decision possibility is. In mathematical terms, preferable means a better objective function value. But what does "preferable" mean?

Figure 2.2: *A swap.*

If real costs are associated with each decision possibility, the objective function is a measure of how *profitable* the revised flight schedule is. However, if costs that reflect flight control principles are used, the objective function is a measure of how *preferable* the revised flight schedule is – given these principles.

The most common approach to quantifying decision costs is to estimate the *real* costs associated with each decision possibilities. As mentioned, 3 decision possibilities exist when solving DARP, namely delaying, cancelling and swapping. An estimate of the real costs associated with each of these include:

- Delay costs.
  - ill-will from customers.
  - missed down-line flights.
  - passengers leaving for other flights undertaken by different airlines.
  - compensations to passengers (hotel stay, discounts, etc.).
  - crew planning issues.
- Swap costs.
  - redirecting passengers to new gates.
  - ill-will from customers as a consequence of changing gates (i.e. a longer walk out to the gate).
  - redirecting luggage, supplies, crew, etc. to aircraft different from those planned.

- Cancellation costs.
  - The costs of cancellations are similar to those of delays, but will often be larger, i.e. there are often more costly ramifications associated with a cancellation than with a delay.

The abovementioned costs are only some of the more important ones to consider when calculating the actual operating costs of an airline. Most of them will be difficult, if not impossible to estimate.

It may be futile to base an objective function on *real* operating costs. Instead, this thesis will focus on quantifying the basic principles that flight controllers adhere to.

To that end, the principles are:

- Delays are preferable to cancellations.
- Some flights have a higher priority than others. The flights that are cancelled or delayed have to reflect this.
- Reassignments of aircraft to flights in the immediate future are not preferable if the disruptions of the schedule can be resolved by making reassignments further down the schedule. For example, last minute changes cause great inconvenience to both passengers and crew because of the lack of time to amend plans.
- At the end of a working day, it is preferable for the number of aircraft at each airport to at least equal the number of aircraft needed at that particular airport the following morning.
- Some specific aircraft (identified by their tail number) must end at a certain airport for scheduled maintenance.
- There are many considerations associated with each individual aircraft that are very difficult to consider properly in any DAR solution method. These include maintenance considerations, whether reverse thrusters may be needed at the destination airport, etc. The point is that a revised and feasible schedule should be found by making the least possible number of reassignments.

To accommodate these principles when solving DARP, costs are specifically designed to reflect the principles and not actual costs.

2.2 The Integrated Crew and Aircraft Recovery Problem

The Integrated Crew and Aircraft Recovery Problem ICARP is a fusion of the Dedicated Crew Recovery Problem (DCRP) and DARP. DCRP is also a more complex problem that DARP for mainly 2 reasons: (i) There are more crew members than aircraft, hence making the DCRP a large-scale problem; (ii) There are a vast number of restrictions on crew assignments. Naturally this makes ICARP a very large problem to solve. The definition of ICARP is given in figure 2.3.

In addition to DARP (figure 2.1), the Integrated Crew and Aircraft Recovery Problem consists of:

- delaying crew
- placing crew on standby
- using **standby crew**
- swapping **crew to flight assignments**
- **dead-heading** standby crew to flights at other airports

in order to create a more preferable revised flight schedule.

Figure 2.3: *Definition of DARP.*

Again, the definition given here contains several concepts that are important to define:

Delaying crew: If a flight has been delayed, the crew assigned to this flight will be delayed as well.

Placing crew on standby: If flights are cancelled, the crew assigned to these can be placed on standby. Crew on standby act as reserve and can in principle be used on any other flight.

Swapping: The concept of swapping crew to flight assignments is exactly like swapping aircraft illustrated in figure 2.2.

Dead-heading crew: Unlike aircraft, crews are sometimes transported to where they are needed. Dead-heading means transporting crew as passengers to flights at other airports where a **crew shortage** exists.

It also includes transporting crew members to their **base station** if they end a **crew pairing** away from home.

2.2.1 Decision Costs

The decision possibilities above must also be associated with costs. Again these costs will reflect certain basic flight control principles. Unlike DARP however, these principles are not very well defined for crew members. Airlines, of course, have operating principles, but there are typically many and their ranking is difficult to ascertain. Consequently, the following set of principles cannot be viewed as factual, but an interpretation of some known flight control principles related to crew:

- As many flights as possible should have the necessary crew assigned.
- **Crew assignments** should not cause flight delays.
- Swapping crew to flight assignments are preferable to using standby crew.
- When swapping crew or using standby crew, some crews are more preferable than others.
- Restrictions on crew (e.g. minimum **turn-around time**) are not all absolute. Among these, some have a higher priority than others.
- Just like DARP, it is a high priority that the number of changes are kept to a minimum.

The listed principles are generalizations of the numerous specific priorities that flight controllers use. They vary from airline to airline which make them difficult to define. A further description with examples is given in chapter 8 where a possible ICARP solution method is described.

Chapter 3

Thesis Statement

The purpose of this thesis is to:

- find and describe a solution method to solve the Dedicated Aircraft Recovery Problem (DARP).
- implement and test this solution method.
- describe a solution method to solve the Integrated Crew and Aircraft Recovery Problem (ICARP) using the results from solving DARP.

Chapter 4

Literature Review

This chapter will discuss relevant literature pertaining to disruption management in the airline industry. Literature on dedicated aircraft recovery is covered first followed by literature on integrated crew and aircraft recovery.

4.1 Literature Review on Dedicated Aircraft Recovery

Early work by **Teodorović and Guberinić (1984)** resolves DARP by using a branch-and-bound procedure in the search of an optimal solution that minimizes the total passenger delay. The work was not documented on general test cases of a realistic size. A single, very simple example was given which consisted of 8 flights. It took more than 300 hours to find an optimal solution.

In **Teodorović and Stojković (1990)** the problem is solved by formulating a model with two objectives using lexicographic[1] optimization. The primary objective is to maximize the number of flights flown and the secondary objective is to minimize the total passenger delay. A greedy heuristic is used to select the order in which **flight links** are created for each

[1]Lexicographic refers to mathematical optimization where 2 or more prioritized objective functions are used.

aircraft. These links are made by solving a shortest path problem for each aircraft where the arc costs reflect the primary and secondary objectives. The model is highly sensitive to how the objective functions are ranked.

Jarrah et al. (1993) develop two separate mathematical models which are solved as minimum cost network flow problems. One model solves the problem by delaying the necessary flights to obtain a solution and the other by finding a set of flights to cancel. The models assume that a so-called dis-utility can be assigned to each flight in order to reflect lost revenue if the flight is cancelled. It is also assumed that the dis-utility of delaying a flight is assessable. This assessment is based on the number of passengers affected, down-line effects, crew and maintenance considerations. The minimum cost network flow problems are solved by Busacker-Gowen's dual algorithm.

The models developed by Jarrah et al. (1993) do not allow for a trade-off between cancelling and delaying a given flight in a combined decision and it does not allow for substitution of aircraft of different types either.

Yan and Yang (1996) set up the Basic Schedule Perturbation Model (BSPM) which is designed to minimize the period in which schedule disruptions exist and maximize the profit simultaneously. A time-space network is used to represent the BSPM and the model is formulated mathematically as a minimum cost network flow problem with side constraints. The model considers delays, cancellations and **ferrying aircraft** simultaneously.

BSPM is solved using a Lagrangian relaxation with a sub-gradient method for a near-optimal solution in order to avoid long computation times. Problem instances were solved where a single aircraft was not available for a whole week due to maintenance requirements.

Cao and Kanafani (1997) combine the possibility of making delays and cancellations in a single model by extending the work of Jarrah et al. (1993). The authors formulate a quadratic binary programming model which maximizes the profit of the operation while taking into consideration both delay costs, costs for substituting aircraft and penalties for cancelling flights. The authors also describe a revised Linear Programming Approximation algorithm to solve the model in which an integral solution is obtained if the approximation algorithm converges.

Argüello et al. (1998) describe a Time Band Approximation model (TBA) that they solve heuristically. They justify using a heuristic approach by showing the difficulties involved in two exact approaches: a resource assignment model, that treats aircraft and cancellations as resources, and a multi-commodity network flow model, where the commodities are aircraft and cancellations. These models are assumed so difficult to solve that an optimal solution cannot be found, thus the development of the approximation model. The TBA model is solved using a Greedy Randomized Adaptive Search Procedure (GRASP). Experimental results on very small flight schedules are shown to illustrate how the heuristic works.

4.2 Literature Review on Integrated Crew and Aircraft Recovery

ICARP was first considered by **Teodorović and Stojković (1995)** where they extend their previous work (see Teodorović and Stojković (1990)) to include crew considerations. Teodorović and Stojković (1995) used a lexicographic approach again and found that rotating crew before aircraft yielded the best results. The optimization of the different resources (crew, aircraft) was done sequentially and each resource was only considered once.

In his Ph.D. thesis **Lettovsky (1997)** develops a mathematical model to solve ICARP. The model is intractable for anything other than small problems, so he suggests solving the problem in a decomposition scheme. The master problem (the Schedule Recovery Model) provides a cancellation and delay plan that satisfy imposed landing restrictions and assigns equipment types. The subproblems can now be solved independently given a solution to the master problem.

Three subproblems are formulated. The first one is the Aircraft Recovery Model (ARM) which generates new flight links that satisfy maintenance requirements. The second one is the Crew Recovery Model (CRM) which generates a new crew schedule that complies with all safety and union regulations. The third and last subproblem is the Passenger Flow Model (PFM) in which new passenger itineraries are generated which comply to seat availability for each flight.

The solution algorithm applies Benders' Decomposition algorithm to a mixed-integer programming formulation of the master problem. In the thesis only the CRM is implemented and tested in order to show the validity of the decomposition scheme. The method appears effective although more computational experiments are needed to demonstrate that larger problems can be solved within a reasonable time frame.

A technical report by **Stojković and Soumis (2000a)** proposes an optimization approach that simultaneously includes crew and aircraft considerations. The objective function aims to minimize changes in the original flight schedule including departure times, aircraft to flight assignments and crew pairings. The model considers **crew types** and **aircraft types** separately to reduce the problem size. This is possible because it is assumed that different types of crew and aircraft cannot be interchanged.

The problem is formulated as an integer nonlinear multi-commodity minimum cost network flow problem with additional constraints. The model is solved by a column generation scheme and a specialized branch-and-bound technique. The subproblems are represented as shortest path problems with time windows and linear costs on flow and time variables. A dynamic programming algorithm for acyclic networks is used to solve the subproblems. Reasonable computational results are shown for some hypothetical examples.

In **Stojković and Soumis (2000b)** the authors elaborate further on their previous work. Here they have added more crew types, features pertaining to passenger connections and aircraft maintenance. The solution methods and algorithms used to solve the problem are the same as the approach used in Stojković and Soumis (2000a).

Again hypothetical problem instances are used for the computational experiments. The results show that extensive computer resources are needed to resolve these instances and the response times are considerably longer.

4.3 Comments on Literature

Generally, methods to solve DARP and ICARP do not seem to be effective enough to be used in real-life. Most approaches involve mathematical models that are impossible to solve efficiently at present. For this reason perhaps, a lot of the research is focused on finding solution methods, e.g. decomposition, that will change this. Not a lot of literature seems focused on developing better conceptual models that ultimately can result in models that are more straightforward to solve.

DARP is a much simpler problem than ICARP. This is probably the reason why literature on DARP seems to have come closer to effective models. This thesis will therefore initially focus on DARP. Particularly 3 articles seem promising in this respect: The heuristic approach by Argüello et al. (1998), the delay and cancellation models by Jarrah et al. (1993) and their extension by Cao and Kanafani (1997).

4.4 Argüello et al. (1998)

In Argüello et al. (1998) an approximation model is developed to solve DARP. The objective of this problem is to find a least cost response to disruptions in the flight schedule.

The Time-Band Approximation Model consists of two phases: (i) the construction of a time-based network in which the planning horizon is partitioned into so-called time bands and (ii) the development of the associated mathematical representation. The following information is assumed given before the construction of the network:

1. Flight data containing the flight identifier, origin, destination, scheduled times of departures and arrivals, expected duration, and a delay cost function.
2. List of aircraft availability including time and location of the aircraft.
3. List of airport data containing an airport identifier and **airport curfew** times.

4.4.1 Description of the Network Representation

The *time bands* are segments of the planning horizon. Within these bands all activities are aggregated to occur in a single node. In other words, the time bands are represented as nodes in the network and arrivals and departures – so-called activities – are represented as arcs either going into or commencing from the node. The network representation is shown in figure 4.1.

This figure illustrates the time band network at a given airport. The turnaround time is assumed to be half an hour for all aircraft and the airport's departure curfew is set to midnight. The time segments are defined to be one hour.

There are 2 types of nodes. The time segments are represented by airport-time nodes 1, 2, and 3 and node 4, which is an airport-sink node. The

Figure 4.1: *Network representation used in the Time-Band Approximation Model.*

airport-time nodes have out-bound arcs that connect to other airports. The number of these arcs for each airport-time node corresponds to the total number of flights scheduled to depart the airport in the considered time horizon. This represents the availability of each flight for each time segment.

If a node has no incoming arcs, such as node 1 in figure 4.1, it means that one or more aircraft at the airport are already available in that time segment. There are also two out-bound arcs from node 1 indicating that two flights are scheduled to depart from this airport after 21.00. Node 1 also has an out-bound arc that terminates in node 4. If an aircraft from node 1 is assigned to this arc, it means that it will terminate at the airport. Nodes 2 and 3 have two and one in-bound flight arcs respectively. These arcs indicate that it is possible for two flights to arrive between 22.00 and 23.00 and for one flight to arrive after 23.00 hours.

It should be clear from this description of the network representation that it is not possible to determine the number of aircraft available at a given time segment by simple inspection. Similarly, whether or not aircraft termination requirements are met cannot be determined by inspection.

4.4.2 Formulation of the Mathematical Model

A mathematical model is based on the time band network. This model is used to obtain tight lower bounds when the delay cost estimates are approximate due to wide time segments.

F	$=$	set of arcs representing flights.
f	$=$	index for flight arcs.
I	$=$	set of airport-time nodes.
i	$=$	index for airport-time nodes.
J	$=$	set of airport-sink nodes.
j	$=$	index for airport-sink nodes. This is also used more generally to index airport-time nodes.
$G(i)$	$=$	set of flights originating at airport-time node i.
$L(i)$	$=$	set of flights terminating at airport-time node i.
$H(f,i)$	$=$	set of destination nodes for flight arc f originating from airport-time node i.
$M(f,i)$	$=$	set of origination nodes for flight arc f terminating at airport-time node i.
$P(f)$	$=$	set of airport-time nodes from which flight arc f originates.
$Q(f)$	$=$	set of airport-time nodes at the airport from which flight arc f originates containing the airport-sink node j, i.e. $P(f)$ is a subset of $Q(f)$.
a_i	$=$	number of aircraft that become available at airport-time node i.
c_f	$=$	the cost of cancelling flight f.
d_{ij}^f	$=$	delay cost of flight f from airport-time node i to j.
h_i	$=$	number of aircraft required to terminate at airport-sink node i.

The decision variables are:

x_{ij}^f	=	amount of aircraft flow for flight f from airport-time node i to j.
y_f	=	cancellation indicator for flight f.
z_i	=	amount of aircraft flow from airport-time node i to the airport-sink node at the same airport.

Using the notation above the model is formulated as follows:

$$\min \sum_{f \in F} \sum_{i \in P(f)} \sum_{j \in H(f,i)} d_{ij}^f x_{ij}^f + \sum_{f \in F} c_f y_f \tag{4.1}$$

subject to:

$$\sum_{i \in P(f)} \sum_{j \in H(f,i)} x_{ij}^f + y_f = 1 \,, \quad \forall\, f \in F \tag{4.2}$$

$$\sum_{f \in G(i)} \sum_{j \in H(f,i)} x_{ij}^f + z_i - \sum_{f \in L(i)} \sum_{j \in M(f,i)} x_{ji}^f = a_i \,, \quad \forall\, i \in I \tag{4.3}$$

$$\sum_{f \in L(i)} \sum_{j \in M(f,i)} x_{ij}^f + \sum_{j \in Q(f)} z_j = h_i \,, \quad \forall\, i \in J \tag{4.4}$$

$$x_{ij}^f \in \{0,1\} \,, \quad \forall f \in F \ \wedge \ j \in H(f,i) \wedge \ i \in I \tag{4.5}$$

$$y_f \in \{0,1\} \,, \quad \forall\, f \in F \tag{4.6}$$

$$z_i \in \mathcal{Z}_+ \,, \quad \forall\, i \in I \tag{4.7}$$

This mathematical model is derived from the time-band network and is in itself a minimum cost network flow with side constraints. The objective function 4.1 minimizes costs stemming from delay and cancellations. Equation 4.2 ensures that all flights are covered, i.e. either the flight must be assigned to an aircraft or else the flight is cancelled. Equations 4.3 and 4.4 are flow conservation constraints. The first one ensures that flow at an airport-time node equals the number of aircraft available at that particular node while the second enforces the **aircraft balance** by requiring that the flow into the airport-sink nodes equals the number of aircraft required to terminate at that particular airport. Equations 4.5, 4.6 and 4.7 ensure that the decision variables are integral: x_{ij}^f and y_f are binary and z_i is a non-negative integer.

The model can be extended to include substitution of aircraft of different types. This extension would change the model into a multi-commodity network flow problem. The model can also be extended to allow ferrying of surplus aircraft. This would increase the number of arcs in the network along with the complexity of the problem. These arcs are assigned a cost that corresponds to the cost of ferrying aircraft.

4.4.3 The Heuristic Design

The mathematical model just described cannot be solved efficiently if reasonably small time bands are used. Argüello et al. (1998) therefore uses a modified GRASP heuristic to solve DARP.

In the GRASP-heuristic used to solve the problem there is no construction phase. Instead, the elements usually employed in the construction phase, i.e. the greedy evaluation function, randomization, and adaptive updating, are now used during the local search. There are 7 main elements to the heuristic and these are:

Initial Solution: A feasible solution forms a starting point for the heuristic. This solution can be obtained by finding the aircraft struck out by disruptions and cancelling the flights they originally were assigned to.

Neighborhood Definition: Neighbor solutions are generated from pairs of flight links in the current best solution. The definition will be described more thoroughly below.

Set of Best Neighbors: During the local search neighbors that compare favorably are kept in a candidate set. The cardinality of the candidate set is controlled arbitrarily.

Evaluation Function: A greedy evaluation function is used to compare the solutions. The marginal costs are calculated and the degree to which the solutions in the candidate set satisfy different restrictions is evaluated. Such restrictions include airport curfews.

Random Selection: From the candidate list a solution is elected randomly.

Adaptive Updating: The best solution found so far is used as a starting point for a new iteration.

Stopping Criterion: There is no mention of the concrete criterion used in the heuristic. Several criteria could be used, e.g. the number of iterations, inadequate improvement in the solution, or a time limit.

Solutions with regard to the heuristic consist of flight links for all the aircraft in the schedule, cancellation links that corresponds to the cancelled flights, and null links that indicate available surplus aircraft. The latter link type is considered as another type of flight link in the following.

The definition of the neighborhood is hence:

> *All feasible solutions that can be generated from a pair of flight links or a flight and cancellation link pair.*

Neighbor solutions are generated using two methods:

1. Flight link augmentation
2. Partial link exchange

In figure 4.2 – taken from Argüello et al. (1998) – a pair of flight links will be used to illustrate how these methods are applied.

Figure 4.2: *Example of two flight links.*

In flight link augmentation, a flight or sequence of flights in the source link are removed and inserted in the target link. The sequence removed from the source link can be inserted before, within, or after the target link. For example, flights 6 and 7 could be inserted after flight 2 and before flight 3 in the target link.

In a partial link exchange, a sequence of flights in the source link is exchanged with a sequence of flights in the target link. Exchanges are performed that either maintain the original final link destinations or swap these. When the original final destinations are maintained, only sequences that have the same origins and destinations are exchanged. For example, flights 2, 3, and 4 can be exchanged with flights 7 and 8. When final destinations are exchanged, both sequences include the final flights in the links. For example, flights 9 and 10 can be exchanged with flight 5 in the target link.

Bounds are generated using the LP relaxation of the TBA model. Feasible solutions generated by the heuristic are compared to these bounds, thus providing a methodology to obtain provably good feasible solutions.

4.5 Jarrah et al. (1993)

Jarrah et al. (1993) introduce two network models to solve the problem of aircraft shortages at an airport. The first model introduced is the Delay Model. It solves the DARP by delaying flights until the shortages are fixed. It allows **swapping** aircraft among flights as well as the use of **surplus aircraft** available either at the airport in question or through ferrying from other airports. The delay model is only able to consider one aircraft shortage and one airport at a time.

The second model is the Cancellation Model which solves DARP by providing an optimal set of flight cancellations. This model also makes use of surplus aircraft and swapping aircraft among flights. Swapping is only allowed if neither aircraft to flight assignment causes a delay. The cancellation model is able to consider all airports simultaneously.

4.5.1 The Delay Model

The Delay Model can be viewed as a network of nodes representing either aircraft or flights, and arcs that represent possible assignments. A minimum cost flow model is applied to this network to provide the optimal aircraft to flight assignments. An example of such a network is shown in figure 4.3.

The vertical axis depicts the hours of the day. Nodes on the left are called aircraft nodes, because they represent aircraft. These nodes are placed at the point in time when the aircraft is ready to depart. The nodes immediately to the right of the aircraft nodes are flight nodes and they

Figure 4.3: *An example of a network of flight-to-aircraft assignments at an airport.*

represent scheduled departures of flights. They are also placed at the point in time when the flight is scheduled to depart.

An arc connecting an aircraft node n to a flight node n' represents an original aircraft to flight assignment. In the example, which is taken from Jarrah et al. (1993), an aircraft shortage at time 14:30 occurs because aircraft 3 will need maintenance. This shortage is represented by a supply of unity at node 3 (the supply being represented by $\triangleright\circ$).

A recovery node, R, is placed at time 17:30 with a demand of less than or equal to 1 (the demand being represented by $\circ\triangleright$). The node is placed at the point in time when the aircraft is repaired and available for a new assignment. Each of the flight nodes are connected to the recovery node to indicate that the repaired aircraft can undertake any of these flights.

A surplus node, S, is placed at the point in time when either the surplus aircraft at the airport is ready or a surplus aircraft ferried from another airport can be ready to undertake a flight. Arcs connect each of the flights to the surplus node. Any other surplus aircraft can be modelled in the same way. The surplus nodes all have a demand of less than or equal to 1. There is also the general restriction, that the accumulated demand at the surplus and recovery nodes has to equal 1.

A backward arc is used to connect a flight node to an aircraft node if the aircraft is different from the aircraft originally scheduled to take the flight. Also the aircraft should be different from the aircraft associated with the shortage. Backward arcs emanating from flight node 3 are shown in figure 4.3. Similar arcs emanate from the other flight nodes, but these are not shown in order to avoid congesting the figure.

The minimum cost flow problem for the described network is then solved. If there is a flow of unity (the upper limit on the flow capacity) in one of the backward arcs, it means that the flight at the tail of the arc is *assigned* to the aircraft at its head. If the backward arc points upward, then no delay is involved since the departure time of the flight is later than the time at which the aircraft is ready to depart. The cost associated with an upward arc is therefore only that of swapping aircraft among the flights. On the other hand, if an *assignment* is made using a backward arc that points downward, then it will incur a delay. Consequently, the cost associated with a downward arc includes both swap and delay costs.

The costs just described are not described in detail anywhere in the literature. A description of a recursive cost estimate function is given that takes into account delays, passengers leaving for other airlines, ill-will and cost of missed connections. All these factors incur costs proportional to the delay in minutes. However, there is no explanation of how these factors are calculated to reflect actual costs.

The arcs connecting the flight nodes with the surplus node have an associated cost that includes swapping and possibly ferrying costs. Similarly, the arcs connecting the flight nodes with the recovery node have an associated cost proportional to the implied delay of the flight. Forward arcs that connect aircraft with flights have no cost.

An example of a possible solution to the minimum cost network in figure 4.3 is shown in figure 4.4. A flow of unity follows the path $3 \rightarrow 3' \rightarrow 4 \rightarrow 4' \rightarrow$ *recovery node*. This means that flight $3'$ is flown by aircraft 4 and flight $4'$ is flown by the recovery aircraft. Notice that the recovery aircraft really is aircraft 3, which was delayed.

Without going into detail, if the delay model were applied to multiple airports, the order in which these airports were considered would become

Figure 4.4: *One possible solution for the delay model shown in figure 4.3*

very important. As will be discussed in section 4.6, Cao and Kanafani (1997) also attempt to solve this problem.

4.5.2 The Cancellation Model

When representing the Cancellation Model, multiple airports are considered. An example also taken from Jarrah et al. (1993) is shown in figure 4.5.

In addition to the representation used in the delay model, arcs are used to connect nodes in different airports. The cost of these arcs is the revenue that would be lost if the flight represented by the tail of the arc were to be cancelled. The only backward arcs allowed at each airport are those that involve no delays (e.g. $9' \rightarrow 8$). This underlying network allows the use of a model, which determines an optimal set of flight cancellations. Both the underlying network and the cancellation model described are very similar to that used in Cao and Kanafani (1997). The principal difference is that in the cancellation model, aircraft cannot be assigned to flights if the assignment incurs a delay. The reason for this again has to do with the order in which the airports are considered; if delays are allowed, down-

Figure 4.5: *An example of a network of aircraft-flight assignments at multiple airports.*

line flights are affected and neither the delay nor the cancellation model described in Jarrah et al. (1993) are able to take this into account.

4.6 Cao and Kanafani (1997)

In Cao and Kanafani (1997) a model is presented which potentially describes an easy way to represent an airline flight schedule for the purposes of making decisions in a disruption situation. The model has two major advantages compared with Jarrah et al. (1993): (i) it considers delays and cancellations simultaneously; (ii) it considers the entire network of airports as an inseparable system. The model is referred to as the Combined Cancellation and Delay Model (CCD)

4.6.1 Graphical Representation used in the CCD Model

The CCD model is in many ways adapted from Jarrah et al. (1993) and the graphical representation is similar. In figure 4.6 an example of a network used in the CCD model is shown.

Figure 4.6: *An example of a network as it is used by Cao and Kanafani (1997).*

Surplus or recovered aircraft are not considered explicitly. Instead such aircraft are represented by a node placed at an airport and time when the aircraft are ready to depart. To distinguish between the nodes, an aircraft node is indexed by a and a flight node by f. Notice that the values of the indexes equals each other for all assignments in the original schedule (i.e. $a = f$).

A term **flight link** is used to describe the sequence of flights performed by one aircraft during the considered time period. In figure 4.6 an example of a flight link is the sequence $a - f \rightarrow a' - f' \rightarrow a'' - f'' = 1 - 1 \rightarrow 5 - 5 \rightarrow 8 - 8$. A connection between a flight and an aircraft at different airports is referred to as a **flight leg**, e.g. the arc between flight node 1 and aircraft node 5 in figure 4.6. The legs in the original schedule are fixed – only the aircraft assignments are changed.

4.6.2 Mathematical Description of the CCD Model

Based on the flight representation just described, the CCD can be formulated. First the variables and constants involved are described:

A	$=$	set of nodes representing aircraft.
a	$=$	index for aircraft nodes.
F	$=$	set of nodes representing flights.
f	$=$	index for flight nodes.
F_a	$=$	subset of F consisting of candidate flights considered for aircraft a. If aircraft a is delayed beyond the time horizon, F_a is set to empty. In figure 4.6, F_a could reasonably consist of flights $\{1, 2, 3\}$ for aircraft $a = 1$.
A_f	$=$	subset of A consisting of candidate aircraft considered for flight f. In figure 4.6, A_f could reasonably consist of aircraft $\{4, 5, 6\}$ for flight $f = 5$.
A_1	$=$	subset of A consisting of aircraft at the first airports of all flight links, during the considered period, according to original schedule. In figure 4.6, subset A_1 consists of $\{1, 2, 3, 4, 9\}$.
$L(f, a')$	$=$	set of flight legs in the schedule. In figure 4.6 $L(1, 5) = 1$, while $L(1, 8) = 0$ or $L(2, 5) = 0$.
r_f	$=$	the revenue of flight f.
s_{af}	$=$	the swapping cost of assigning aircraft a to flight f, including the cost caused by changing the gate, informing the crews and passengers of the change, ferrying surplus aircraft, etc., but not including delay cost.
d_{af}	$=$	delay cost at the first airport in each link (see later on in this section).
$d^{af}_{a'f'}$	$=$	delay cost at the second airport in each link (see later on in this section).
$d^{af}_{a'f'}(a'')$	$=$	accumulated delay cost at the third airport in each link (see later on in this section).

The decision variables are:

$$x_{af} = \begin{cases} 1 & \text{if aircraft } a \text{ is assigned to flight } f \\ 0 & \text{otherwise} \end{cases}$$

Constraints in the CCD Model

Cao and Kanafani (1997) list only two constraints in their model. These constraints are:

$$\sum_{f \in F_a} x_{af} \leq 1, \qquad \forall\, a \in A \tag{4.8}$$

$$\sum_{a \in A_f} x_{af} \geq \sum_{f' \in F_{a'}} x_{a'f'}, \qquad \forall\, L(f,a') = 1 \tag{4.9}$$

Equation 4.8 ensures that only one flight is assigned to each aircraft. Equation 4.9 ensures that a given aircraft is only assigned to a flight if the preceding aircraft to flight assignment is made. In Cao and Kanafani (1997) they claim that a constraint ensuring that only one aircraft is assigned to each flight is redundant: If equations 4.8 and 4.9 are added, the resulting constraint will ensure exactly that. However, this is *not* true, because equations 4.8 and 4.9 do not cover the same indexes. Thus a third constraint has to be introduced:

$$\sum_{a \in A_f} x_{af} \leq 1, \qquad \forall\, f \in F \tag{4.10}$$

However, as will be shown in section 5.5.1, these constraints are still not sufficient to ensure a feasible solution.

Objective Function of the CCD Model

A very important part of the CCD model is the intricate objective function. It consists of 5 parts named $\varphi_1, \ldots, \varphi_5$.

Part 1: The revenue of all flights:

$$\varphi_1 = \sum_{f \in F} \left(r_f \sum_{a \in A_f} x_{af} \right) \tag{4.11}$$

Part 2: Swapping costs

The swapping cost is the cost of assigning an aircraft to a flight different from that specified in the original flight schedule:

$$\varphi_2 = \sum_{f \in F} \sum_{a \in A_f} s_{af} x_{af} \tag{4.12}$$

There is no swap cost associated with keeping the original assignment (i.e. $s_{af} = 0$).

Next follows the costs associated with delays. These costs have been divided into 3 different parts, namely the cost associated solely with delays at the first airport in each flight link, and likewise for the second an third airport. The delay cost at the third airport is an accumulated delay cost of delays at the third and all down-line airports in the flight link.

Part 3: The delay cost associated with aircraft in subset A_1:

$$\varphi_3 = \sum_{a \in A_1} \sum_{f \in F_a} d_{af} x_{af} \tag{4.13}$$

The delay of aircraft at their first airport is a simple calculation of the delay incurred should aircraft a be assigned to flight f. In other words, the **ready-time** of aircraft $a \in A_1$ is compared to the originally scheduled departure times of flights $f \in F_a$. The delay cost d_{af} in equation 4.13 is proportional to this delay and is calculated before running the model.

Part 4: The delay cost associated with aircraft at their second airports:

$$\varphi_4 = \sum_{a \in A_1} \sum_{f \in F_a} \sum_{L(f,a')=1} \sum_{f' \in F_{a'}} d^{af}_{a'f'} x_{af} x_{a'f'} \tag{4.14}$$

Calculating the delay at airport 2 in each link is a little more complicated than delays at airport 1, because these depend on the assignments made previously in the flight schedule. In figure 4.7, an assignment scheme is shown, and the corresponding delays are calculated in table 4.1. The assignment scheme and the corresponding delay costs ($d^{af}_{a'f'}$) are based on a delay t_0 of aircraft 1.

Figure 4.7: *Assignment scheme*

Assignments	$1 \to 1$ and $3 \to 3$	$1 \to 1$ and $3 \to 4$	$2 \to 1$ and $3 \to 3$	$2 \to 1$ and $3 \to 4$
Delay	t_0	$\max\{0, t_0 - t_2\}$	t_1	$\max\{0, t_1 - t_2\}$

Table 4.1: *Delay time caused by aircraft 3*

According to equation 4.14, the delay cost associated with aircraft 3 at its second airport becomes:

$$\varphi_4 = d^{11}_{33} x_{11} x_{33} + d^{11}_{34} x_{11} x_{34} + d^{21}_{33} x_{21} x_{33} + d^{21}_{34} x_{21} x_{34}$$

The delay cost is again proportional to delay resulting from the assignments made.

Part 5: The delay cost associated with aircraft at their third and down-line airports

$$\varphi_5 = \sum_{a \in A_1} \sum_{f \in F_a} \sum_{L(f,a')=1} \sum_{f' \in F_{a'}} \sum_{L(f',a'')=1} d^{af}_{a'f'}(a'') x_{af} x_{a'f'} \tag{4.15}$$

The delay of an aircraft at its third airport exactly corresponds to the delay of the corresponding flight at the second airport. However, the associated delay cost $d^{af}_{a'f'}(a'')$ at a third airport, is an estimate of the costs associated with the flight taken by the delayed aircraft at its 3rd airport and its following down-line flights. This estimate is based on the delay model described in Jarrah et al. (1993), which was discussed earlier in this chapter (see section 4.5.1).

Given a delay at airport 3 in figure 4.8, an optimal reassignment scheme is found by solving the delay model. The corresponding delay cost is calculated based on the assignments found. The following should be noted: (i) it is assumed that all other aircraft at airport 3 are on time; (ii) the surplus node shown in figure 4.8 is really just another aircraft node, but the swap cost of using this aircraft reflects possible ferrying costs, readying costs, etc.; (iii) the sum of demands at the surplus and recovery nodes has to equal 1.

Figure 4.8: *Delay of aircraft at its third airport*

To get an idea of the reassignment scheme, refer to figure 4.8 where aircraft 1 is delayed by t_0. If the assignment x_{11} and x_{33} is made despite the delay to aircraft 1, aircraft 5 at airport 3 will be delayed. Given this delay, the minimum cost network constructed at airport 3 is solved placing a supply of 1 at aircraft 5 and a demand of 1 or less at the recovery and surplus nodes.

The result of solving this minimum cost network is a sequence of assignments. For example $(5-5) \rightarrow (5-4) \rightarrow (4-4) \rightarrow (4-$ surplus aircraft$)$ means that flight 5 will be flown by aircraft 4 and flight 4 will be flown by the surplus aircraft. This sequence of assignments is based on the delays incurred and the swap costs.

In the example just given, the aircraft flying the link $(1-1) \rightarrow (3-3) \rightarrow (5-)$ no longer has any assignments once it reaches airport 3, given the reassignment scheme just found. Had the solution to the minimum cost network instead been $(5-5) \rightarrow (5-5')$ the original link starting with aircraft 1 would be retained including any legs beyond flight 5.

It is very important to realize, that given the first and second assignments in a flight link, all remaining assignments are determined by the outcome of the delay model applied. If this were not the case, the delay cost at the third station – which is calculated solely on the basis of the first and second assignments – would not reflect the assignments made here.

The CCD model

Below the revised CCD model is presented as a whole:

$$\max \quad \varphi = \varphi_1 - \sum_{i=2}^{5} \varphi_i \tag{4.16}$$

s.t.

$$\sum_{f \in F_a} x_{af} \leq 1, \qquad \forall\, a \in A \tag{4.17}$$

$$\sum_{a \in A_f} x_{af} \leq 1, \qquad \forall\, f \in F \tag{4.18}$$

$$\sum_{a \in A_f} x_{af} \geq \sum_{f' \in F_{a'}} x_{a'f'}, \qquad \forall\, L(f, a') = 1 \tag{4.19}$$

$$x_{af} \in \{0, 1\}, \qquad \forall\, a \in A \wedge \forall\, f \in F \tag{4.20}$$

where $\varphi_1, \ldots, \varphi_5$ are given by equations 4.11, 4.12, 4.13, 4.14 and 4.15.

4.7 Conclusion on Chapter 4

With respect to DARP, the most interesting article is the one by Cao and Kanafani (1997) describing the so-called CCD model. They present a seemingly robust model that can be solved within a reasonable time frame while considering both delays, swaps and cancellations simultaneously. Another article by Argüello et al. (1998) presents an interesting heuristic approach, but its complicated structure makes it less attractive than the CCD model.

ICARP seems to be an illusive problem. There are articles which present interesting theoretical approaches, but they are a long way from solving problems of a realistic size within a reasonable time frame.

Chapter 5

Verification of the CCD Model

This chapter describes the attempt to verify that the CCD model by Cao and Kanafani (1997) works as described. This includes a description of the method of implementation and the problem instances generated.

5.1 Motivation

There are several reasons for trying to verify the results achieved by the CCD model. These reasons are:

- The CCD model combines the possibility of delaying and cancelling aircraft.
- It considers the airport network as an inseparable system.
- The authors ostensibly solve realistically large problem instances within reasonable computational time.
- Assuming that the CCD model works as described, it will presumably form a good foundation for further development.

Another article by Yan and Yang (1996) also presents a model, which seemingly includes the same considerations as the CCD model. However, the problem instances which are solved by Yan and Yang (1996) are small and based on a single disruption that lasts an entire week. This seems less operational than the problem instances solved by Cao and Kanafani (1997); these are large and include delays on 20-30% of the aircraft.

5.2 Creating Problem Instances

Before describing the implementation of the CCD model, a general description of the problem instances and how they are generated will be given. A problem instance in this respect is a flight schedule covering both aircraft and airports and including delays. The integrity of these problem instances is vital, if the validity of the model is to be tested properly.

5.2.1 Cost Design Principles

The cost design is important for the CCD model to work as intended. Likewise, costs determine how realistic the flight schedules found by the CCD model will appear. Below is a description of the principles behind the costs used in the model.

- Revenue
  - The revenue for all flights has been set to \$1000.
- Delays
  - The difference between a flight's original departure time and its actual departure time constitutes the delay.
  - Flights never depart before their original departure time.
  - The delay costs d_{af}, $d_{a'f'}^{af}$ and $d_{a'f'}^{af}(a'')$ are functions of the respective flight revenues and their possible delays measured in minutes.
- Swaps
  - A swap cost s_{af} is used to prioritize swaps.
  - For the sake of evaluating the flight schedules found by the heuristic, the swap cost s_{af} have been set to 0 for all combinations of a and f.
- Cancellations
  - Cancellations are not specifically assigned a cost in the CCD model. However, the revenues of cancelled flights do not contribute to the objective function.

Calculating Delay Costs

The delay costs are proportional to functions of the respective flight revenues and their delays measured in minutes. The functions used to calculate flight delays are defined as follows:

$$d_{af} = DelayInMinutes(a, f) \cdot DF \cdot r_f \tag{5.1}$$

$$d^{af}_{a'f'} = DelayInMinutes(a, f, a', f') \cdot DF \cdot r_f \tag{5.2}$$

Equations 5.1 and 5.2 are calculated for all possible aircraft to flight assignments prior to running the mathematical model. The parameter $DelayInMinutes(a, f)$ used in equation 5.1 is the delay of a flight f as a consequence of aircraft a being assigned to this flight. Likewise, $DelayInMinutes(a, f, a', f')$ is the delay of flight f' as a consequence of 2 consecutive aircraft to flight assignments (see table 4.1). The parameter DF is explained in section 5.2.2.

The delays associated with aircraft at their third and down-line airports ($d^{af}_{a'f'}(a'')$) are more complex. According to Cao and Kanafani (1997), the delay cost is found after solving a minimum cost network like that described in Jarrah et al. (1993). In this implementation a simplified way of dealing with assignments at the third airport in each link was chosen. Firstly, it is assumed that the optimal solution to the minimum cost network always means retaining the original assignments. In figure 4.8 this means that for all possible assignments leading a link to aircraft node 5, the original assignment $(5 - 5)$ is retained. It also means, that flight 5 cannot be cancelled. Naturally, these limitations may exclude the possibility of finding an optimal reassignment scheme. However, initially we aim only to have the model generate a feasible solution. To this end, our limitations are equivalent to that of calculating the optimal solution of a minimum cost network at airport 3 in each link.

5.2.2 Parameter Values

The nature of the problem instances generated to test the CCD model depends on a series of parameters. Below follows a description of the parameters, which change throughout the problem instances. These parameters all have significant impact on the size of the problem.

Number of aircraft: The number of flights which are flown in a generated schedule is roughly proportional to the number of aircraft. This parameter is therefore an important factor in determining the problem size.

Number of airports: The relationship between the number of airports and the number of aircraft determines the **aircraft density** at each airport. The density is defined as the number of aircraft that take off from the airport within a specified time horizon. As will be explained below, the candidate sets are proportional in size to the aircraft density.

Candidate sets F_a: The candidate sets are very important with respect to the size of the solution space. The smaller these sets are, the smaller the solution space. In all the generated problem instances an aircraft can be assigned to any flight leaving from its current airport, excluding those flights which presently are a part of that aircraft's link (see section 5.5.3).

Delay percentage: This parameter indicates the approximate number of aircraft that are delayed. The delay percentage has been set to 20% in all problem instances. Aircraft are always delayed at their first airport. There is a good reason for this approach: If an aircraft is assigned to flight such that no delays should arise, then it is not possible to predict delays that may arise further down the link. For this reason aircraft are never delayed at their second or down-line airports, because information about such delays would only exist if the aircraft had been delayed at its first airport. It is of course assumed that all flights down-line from a delay are delayed as well.

Number of flights: This is not actually a parameter. It is derived after a schedule has been created based on the above parameters.

Aside from the parameters just listed there are parameters which are not changed throughout the problem instances generated to test the CCD model. These are:

Time horizon: The length of a flight link is limited by the time horizon. The time horizon has been set to 420 minutes.

Flying time: It is assumed that all flights have the same flying time. This can easily be changed, but for the sake of simplicity the flying time has been set to 110 minutes for all flights. In this implementation the flying time also includes the turn-around time, i.e. flying time is the time it takes from when the aircraft departs from one airport and until it is ready to depart from the next airport.

Delay factor DF: The delay factor determines how much revenue to deduct from the objective function given a certain delay. More precisely, the delay factor is the percentage of the revenue that is removed per minute delay of a given flight; DF has been set to 1%. In other words, the entire revenue of a flight f is subtracted from the objective value if that flight is delayed for exactly 100 minutes.

The values assigned to the parameters used in the CCD model are not necessarily realistic from a practical point of view. However, their relative size is reasonable from an experimental point of view. The purpose of this chapter is only to test and verify the CCD model, so problem instances are required to have a realistic structure, not necessarily realistic values. To this end, the values assigned are acceptable.

5.3 Implementing the CCD Model

All the problem instances generated to test the CCD model are generated by a *C++*-program, which writes the instances in *GAMS* syntax. *GAMS* is then run to solve the CCD model using *DICOPT*. *DICOPT* is a solver that solves nonlinear discrete problem types, and thus allows a quadratic binary integer problem to be solved, although *DICOPT* only is capable of finding a local optimum, due to the nonlinearity.

The *GAMS* model can be found in appendix A together with the *C++*-program used to generate the model.

In Cao and Kanafani (1997) they repeat the search for a local optimum in order to find better solutions. Initially, the model is tested by only completing *one* local search: If the solution produced is not valid, then the CCD model does not work as described.

Airport 1			Airport 2		
No.	Revenue	Time	No.	Revenue	Time
1	1000	12	7	1000	19
2	1000	32	8	1000	122
3	1000	118	9	1000	142
4	1000	129	10	1000	228
5	1000	232	11	1000	239
6	1000	252			

Table 5.1: *Scheduled departure time and revenue of the flights in sl-2-4-11. Note that all the flights are flown between airport 1 and 2, i.e. flights from airport 2 end in airport 1.*

Flight link link number	Scheduled departure time	Actual ready-ready-time of aircraft	Link
1	12	34	(1-1) → (8-8) → (5-5) →
2	118	118	(3-3) → (10-10) →
3	19	19	(7-7) → (4-4) → (11-11) →
4	32	198	(2-2) → (9-9) → (6-6) →

Table 5.2: *Flight links in the original schedule of sl-2-4-11.*

5.4 Testing the CCD Model

Initially one problem instance *sl-2-4-11* was generated to test the CCD model. The problem instance name indicates that there are 2 airports, 4 aircraft and 11 flights in the problem instance. In table 5.1 the schedule and revenues of the flights in this problem instance are shown. The column "*No.*" refers to both the flight and the aircraft flying it.

In table 5.2, the flight links in the original schedule are shown. Each "*Flight link number*" corresponds to a physical aircraft. The column "*Scheduled departure time*" shows the ready-time of the respective aircraft in the original schedule; "*Actual ready time of aircraft*" shows when each aircraft will actually initiate its link after taking into consideration the imposed delays.

The revised flight schedule is shown in table 5.3. This revised schedule was found by the CCD model when applied to optimize the *sl-2-4-11* problem instance. The revised flight schedule is also shown graphically in figure 5.1.

Flight link number	Actual ready-time of aircraft	Link
1	34	(1-6) →
2	118	(3-3) → (10-10) →
3	19	(7-7) → (4-4) → (11-11) →
4	198	(2-5) →
ghost aircraft	?	(5-2) → (9-9) → (6-1) → (8-8) →

Table 5.3: *Revised flight links found by the CCD model when applied to the sl-2-4-11 problem instance.*

Figure 5.1: *Graphical representation of the revised links shown in table 5.3.*

In figure 5.1, a black aircraft node indicates that an aircraft is located at that particular airport at the beginning of its original link. A white aircraft node indicates that the aircraft is situated at the present airport as a consequence of the flown flight leg. A label a^* indicates that the aircraft is delayed and an upward pointing arc represents an assignment that causes a delay (e.g. $6^* - 1$). All assignments are represented by forward arcs between aircraft and flight nodes. An arc from a to f, where the values of the indexes $a = f$ indicate that the original assignment is maintained in the optimized schedule (e.g. $3 - 3$).

In figure 5.1, the dotted flight link is flown by a *ghost* aircraft: $(5 - 2) \rightarrow (9 - 9) \rightarrow (6 - 1) \rightarrow (8 - 8) \rightarrow (5 - 2) \ldots$. By *ghost* aircraft is meant that no physical aircraft ever initiates the flight link represented by the circuit. For there to exist a real aircraft in any link, it has to include node 1, 2, 3, or 7 in its path (refer to figure 5.1). These nodes are the physical aircraft assigned to each original link. As can be seen, none of these nodes are included in the circuit, hence the term *ghost* aircraft is introduced to describe the phenomenon. It is furthermore interesting to notice that the ghost aircraft absorbs all the delays; none of the *real* aircraft are assigned to flights that cause them to be delayed.

5.5 Errors in the CCD Model

Naturally, a ghost aircraft flying in a circuit like the one in figure 5.1 is illegal. Either there are problems with the implementation of the CCD model or worse – the CCD model simply does not work.

When we discovered that we were unable to make the CCD model work, we initially assumed that something was wrong with our implementation. We then spent a significant amount of time making sure that our implementation strictly followed the model described in Cao and Kanafani (1997). Through this process we discovered several problems that mathematically render the CCD model as described inoperable. These problems are described in the following section. Attempted modifications to the CCD model are described in section 5.6.

5.5.1 Circuits and Ghost Aircraft

The most obvious problem with the revised schedule illustrated in figure 5.1 is the circuit and the ghost aircraft.

First of all, this circuit is allowed because it does not violate any of the constraints in the model presented in Cao and Kanafani (1997). Equations 4.17 and 4.18 in section 4.6 ensure that no more than one aircraft is assigned to a flight and vice versa. These constraints are obviously not violated. Equation 4.19 ensures that an assignment only is made if the preceding assignment also is made. A circuit of course fulfills this criterion perfectly. Thus, the ghost aircraft is not prevented by the constraints in the CCD model.

Secondly, the objective function chooses the ghost aircraft because it is very advantageous. Equation 4.11 described in section 4.6 is the positive part of the objective function that the model will attempt to maximize. Obviously, the more assignments that are made, the greater the revenue. However, in equations 4.13, 4.14, and 4.15 costs as a consequence of assignments are calculated on the basis of only legal links originating in an aircraft node where a physical aircraft exists. Hence, the entire circuit made by the ghost aircraft entails no cost, only revenue.

The fact that the CCD model does not exclude the possibility of circuits is a very significant error. In all the problem instances that were evaluated by the CCD model, this error occurred. As mentioned, modifications to the CCD model that possibly correct this problem are described in section 5.6.

5.5.2 Evaluating Costs at the Third Airport in each Link

Another serious problem with the CCD model arises when delay and cancellation costs are calculated at the third airport in each link. As described in section 4.6, a minimum cost flow problem is used to calculate the delay cost at the third airport in each link. This method of evaluating delay costs was introduced by Jarrah et al. (1993), however, there were several important limitations with the delay model introduced here:

- Only *one* airport is considered at a time.
- Only *one* delayed aircraft can be considered at a time.
- All aircraft other than the delayed aircraft are assumed to be on time.

These 3 limitations are supposedly *not* limitations in the CCD model. However, this cannot be entirely true since the CCD model makes use of the delay model. Following is a more precise explanation of why there is a conflict.

The delay cost – calculated using the delay model – is based on a series of reassignments. It must therefore be assumed that these assignments in fact are made. However reasonable this assumption may be, serious limitations consequently arise .

Figure 5.2: *Flight schedule according to the CCD model.*

In figure 5.2, a normal flight schedule is illustrated. According to the CCD model there is a certain revenue along with possible delay costs associated with this flight schedule. Suppose the CCD model makes a swap like that illustrated in figure 5.3.

Aircraft 6 is now assigned to flight 5 and vice versa. This results in 2 new links: $\alpha = (1-1) \rightarrow (6-5) \rightarrow (10-10)$; $\beta = (5-6) \rightarrow (11-11)$. To evaluate the quality of the new assignments, the revenue/cost of links α and β are calculated. If it is assumed that aircraft 1 is delayed by t_0, then the cost associated with link α according to the CCD model has the following 3 components:

- Delay cost of the assignment $(1-1)$ proportional to t_0.
- Swap cost and delay cost of the assignment $(6-5)$.
- Possible delay and swap cost of the assignment made at the 3^{rd} airport in link α and an estimate of the cost incurred to the down-line flights. The delay cost at the 3^{rd} airport will be based on the solution to the minimum cost flow problem described in section 4.6.

Suppose there is a method of enforcing the assignments on which the delay costs at the third airport are based – despite the limitations of the delay

Figure 5.3: *Revised flight schedule with swap between aircraft 5 and 6. The question mark indicates that this assignment depends on the solution to a minimum cost flow problem.*

Figure 5.4: *Conflicting assignments forced by solution to minimum cost flow problem at the 3rd airport in the link α.*

model. This is where a problem arises and to illustrate why, look at link α and $\gamma = (2-2) \rightarrow (7-7) \rightarrow (12-12)$ both shown in figure 5.4.

The 3^{rd} airport visited by these links are identical (airport 3). Suppose the minimum cost flow problem solved at the 3^{rd} airport for link α yields an optimal solution, where the reassignment $10 - 12$ should be made. This would give rise to a conflict with the assignment $12 - 12$ of link γ. The problem is that both links end up taking the same flight consequently conflicting with a basic restriction in the CCD model (see equation 4.17) that prevents exactly that. The result is that a proper delay cost associated with the 3^{rd} assignment in link α cannot be found in this situation.

The situation just described is infeasible and will, hence, not appear when using the CCD model. However, because the assignments at the 3^{rd} airport in each link solely depend on the previous 2 assignments, link α and γ can never coexist. The reason is that the CCD model will not allow the 2 links to be assigned to different flights at their 3^{rd} airport (airport 3). In other words, large parts of a realistic solution space are cut away making it unlikely that actual optimal solutions can be found.

Notice, there could be another interpretation to the use of the delay model. Suppose the delay model is only used to *estimate* the delay costs at the third airport. In other words, it is no longer assumed that the assignments – on which the delay costs are based – are enforced. In such a situation the assignments at the 3^{rd} airport in each link would be associated with a cost based on the delay model, but the CCD model would be able to make other assignments. This approach is perhaps possible. However, there is a serious risk of not being able to associate appropriate costs with the assignments actually made at the 3^{rd} airport, hence rendering the model inoperable. Certainly nothing is mentioned about this issue in Cao and Kanafani (1997).

5.5.3 Candidate Sets

Very little attention is given to the candidate set in Cao and Kanafani (1997). It is only mentioned that smaller candidate sets yield a smaller solution space. However, the candidate sets are in fact critical in preventing a certain type of circuit from arising. To understand this, refer to figure 5.5.

Figure 5.5 is a small flight schedule. As can be seen, aircraft 1 initiates the link illustrated by a dashed line. This link visits airport 1 twice, and it is in this situation that the candidate set must be modified. If the candidate

Figure 5.5: *Situation where aircraft 1 and 3 cannot be swapped.*

set to aircraft 3 includes the possibility of assigning it to flight 1, then an impossible situation occurs: Aircraft 3 only exists if aircraft 1 has been assigned to flight 1, so naturally aircraft 1 and 3 cannot be swapped. This situation is easily changed by modifying the candidate set of aircraft 1 and 3. However, in other situations aircraft 1 and 3 *can* be swapped. In figure 5.6 a situation is illustrated where this is the case.

Figure 5.6: *Situation where aircraft 1 and 3* ***can*** *be swapped.*

The candidate sets must therefore be dynamic if circuits, like the one shown in figure 5.1, are to be avoided. This is very difficult in mathematical modelling, because the constraints would vary in such a situation. In other words, some change must be made to the CCD model that ensures only feasible assignments.

5.6 Modifications to the CCD Model

In order to possibly rectify the CCD model, modifications have been attempted that would (i) allow only feasible solutions, (ii) retain a limited model complexity.

There are 2 other issues with respect to the modification of the CCD model. Firstly, with regard to the problems with delay costs at the third airport in each link, it is initially assumed that some solution can be found to this problem. The original assignments are simply retained here for the purposes of finding a method of generating feasible solutions. Secondly, it is assumed that the problems with the candidate sets can be solved if new constraints or methods are found, which eliminate the possibility of circuits altogether.

5.6.1 Interpretation of Airports

In the problem instance *sl-2-4-11* there were only 2 airports. This means that each of the links 1, 3, and 4 (see table 5.2) visited the same airport twice as shown in figure 5.7. Possibly, Cao and Kanafani (1997) did not intend it to be possible for the same link to visit one airport more than once because it created the possibility of circuits – despite their own data sets having such repeated visits at the same airport. As a consequence, a problem instance *sl-5-4-11* was created. Each airport in this instance is at most visited by each link once. However, a new circuit with a ghost aircraft arose in the optimal solution spanning 3 different airports. The conclusion is that circuits are not prevented if links only visit the same airport once.

It is possible to interpret airports in still another way. In figure 5.7, the original schedule to the problem instance *sl-2-4-11* is shown. In figure 5.8, the same problem instance is represented differently; airports 3 and 4 have been created to represent airports 1 and 2 respectively. This representation ensures that all arcs point in the same direction.

The idea behind this representation is that no reassignments can result in a circuit provided reassignments are only made within the same airport; for example, candidate flights for aircraft 4 in figure 5.8 are $\{4, 5, 6\}$. If the model described in Cao and Kanafani (1997) is applied to problem instance *sl-2-4-11* represented as shown in figure 5.8, a feasible solution is found.

Figure 5.7: *The original schedule in problem instance sl-2-4-11.*

Figure 5.8: *The original schedule in problem instance sl-2-4-11 with new airport structure.*

The problem with this representation is that good solutions are likely to be excluded because the number of candidate flights for each aircraft is reduced. Flights $\{1, 2, 3\}$ are, for example, no longer candidates for aircraft 4, despite actually being at the same airport. If the candidate sets are returned to their original size (i.e. $A_4 = \{1, 2, 3, 4, 5, 6\}$) while retaining the new airport structure, then the solution illustrated in figure 5.9 is generated.

It is easily observed that a circuit appears in the optimal solution when the original candidate sets are used. The reason for this is, that the aircraft to flight arcs no longer only point in the same direction. If aircraft 4 is allowed to take flight 3 then a backward arc is possible and circuits may

Figure 5.9: *Solution to problem instance sl-2-4-11 with original candidate sets and new airport structure.*

appear.

In conclusion, changing the layout of the airports and the candidate sets does not solve the problems with the CCD model. In order to find feasible solutions, the candidate sets have to be very restrictive. This will very likely prevent optimal solutions from being found. On the other hand, if candidate sets are not restrictive, circuits may appear yielding infeasible solutions. Neither option seems acceptable.

5.6.2 Constraints to Prevent Circuits

Given the time horizon used in both examples *sl-2-4-11* and *sl-5-4-11*, it is reasonable to assume that a link in an optimal solution will not contain more than 4 assignments for the following reason: The number of assignments in each flight link in the original assignment is limited by the time horizon. The starting time of each link is generated randomly and the following sequence of airports visited by the link is also randomly selected. The longest possible link, given the time horizon and the flying time between airports in *sl-2-4-11* and *sl-5-4-11*, would have 3 assignments. Hence, a link in the optimal solution with more than 4 assignments would have significant delays. Furthermore, a mathematical model may not have information to make qualified guesses as to the optimal 5^{th} assignment in a flight link. It is therefore reasonable to assume a maximum link length of 4 assignments in an optimal solution.

A constraint ensuring a maximum length of 4 assignments would eliminate all circuits, because any circuit in this respect has an infinite number of assignments. This constraint would look like this:

$$\begin{aligned} x_{af} + x_{a'f'} + x_{a''f''} + x_{a'''f'''} + x_{a''''f''''} &\leq 4 \\ \forall \ \{(a, f, a', f', a'', f'', a''', f''', a'''', f'''') \ | \\ a \in A, f \in F_a, L(f, a') = 1, f' \in F_{a'}, L(f', a'') = 1, \\ f'' \in F_{a''}, L(f'', a''') = 1, f''' \in F_{a'''}, L(f''', a'''') = 1, \\ f'''' \in F_{a''''}\} \end{aligned} \tag{5.3}$$

To illustrate how this constraint works, refer to the circuit in figure 5.9. The cut which would eliminate this circuit would, according to equation 5.3, look like this:

$$x_{4,3} + x_{10,7} + x_{4,3} + x_{10,7} + x_{4,3} \leq 4 \tag{5.4}$$

In theory this would result in a maximum of approximately n^5 different constraints, where n is the number of flights in the flight schedule. Without dwelling any further on this issue, introducing these constraints would make the model impossible to solve for anything but very limited flight schedules.

5.6.3 Generating Cuts

Imposing constraints to prevent circuits (equation 5.3) results in a model too large to solve. However, the vast majority of these constraints are not going to be violated at any time, so it is reasonable to attempt to solve the model by generating cuts. Hence, the original model given in equations 4.16, 4.17, 4.18, and 4.19 is viewed as a relaxed problem where the restriction given in equation 5.3 has been removed. Below is illustrated the cut generation on the problem instance *sl-2-4-11*.

The original schedule in *sl-2-4-11* is shown in figure 5.7. When this is solved, the solution shown in figure 5.1 is found. The ghost aircraft flying the circuit $(5-2) \rightarrow (9-9) \rightarrow (6-1) \rightarrow (8-8) \rightarrow (5-2)\ldots$ is prevented by the following cut:

$$x_{5,2} + x_{9,9} + x_{6,1} + x_{8,8} + x_{5,2} \leq 4 \tag{5.5}$$

No. of flight link	Actual ready time of aircraft	Link
1	34	(1-2) $\rightarrow$ (9-9) $\rightarrow$ (6-6) $\rightarrow$
2	118	(3-4) $\rightarrow$ (11-11) $\rightarrow$
3	19	(7-8) $\rightarrow$ (5-5)
4	198	Aircraft cancelled
ghost aircraft	?	(4-3) $\rightarrow$ (10-7) $\rightarrow$

Table 5.4: *Optimal flight links after solving the model.*

Figure 5.10: *The solution to sl-2-4-11 according to table 5.4.*

Equation 5.5 is added to the model, which is solved again. The resulting schedule is shown in table 5.4 and illustrated in figure 5.10. As can be seen, the solution is still not feasible because of a circuit.

The ghost aircraft flying the circuit $(4-3) \rightarrow (10-7) \rightarrow (4-3)\dots$ in figure 5.10 is prevented by the cut:

$$x_{4,3} + x_{10,7} + x_{4,3} + x_{10,7} + x_{4,3} \leq 4 \tag{5.6}$$

Equation 5.6 is therefore also added to the model, which again is solved. This time a feasible solution is found with no circuits as shown in figure 5.11.

Figure 5.11: *A feasible solution to sl-2-4-11 found using cuts.*

In conclusion, it is possible to use cuts to find feasible solutions when using the CCD model. The complexity of finding these cuts is $\mathrm{O}(n^2)$ given a solution to the relaxed problem (n is the number of flights). For each flight f, the flight link to which it belongs is traversed to see if this link forms a circuit. The flight link can at most contain n flights, hence it takes $\mathrm{O}(n^2)$ to find a circuit.

In larger problems, a vast number of cuts may have to be generated before a feasible solution is found. It is therefore reasonable to expect, that using cuts will be very time consuming, especially considering that the model is time consuming to solve even when it is relaxed. Further experiments with this method are therefore not undertaken.

5.6.4 Involving Time

If it were possible to keep track of the time, constraints could be made to ensure that the number of aircraft in use at any time never exceeds the actual number of aircraft. In figure 5.1 there are actually 5 aircraft in use despite there only being 4 available. If constraints like the one just mentioned were implemented, perhaps only legal solutions would arise. However, there are at least 2 problems with this approach: (i) The model allows cancellations of flights and even aircraft. There is nothing to prevent the model from cancelling a real aircraft and creating a ghost aircraft that flies around in a circuit, thus satisfying this new constraint; (ii) There is no way to properly define the ready time of a ghost aircraft, hence it would be very difficult to use it in a constraint.

5.7 Conclusion on Chapter 5

The purpose of this chapter was to verify that the CCD model by Cao and Kanafani (1997) worked as described. This has not been possible because the CCD model contains at least 3 errors:

- Circuits are allowed in the solutions found by the CCD model.
- There is no proper way to associate the delay costs at the third airport in each link with the actual assignments made.
- The candidate sets are not defined correctly.

Primarily, we were unable to find a satisfactory solution to the error having to do with circuits. Either the solution simply did not work or the model became too difficult to solve. As a consequence, the problem concerning the candidate sets has not been solved either. We have therefore not bothered trying to find a better method of evaluating the costs at the third airport in each link.

Given the serious errors in the CCD model, it seems strange that no one – especially the authors – has discovered or commented on these errors before. It is possible that the authors overlooked the errors in the model because of the large number of surplus aircraft used in their implementation. These would minimize the effects of the disruptions significantly and possibly eliminate most circuits. However, on closer inspection, there are also errors in the solution given in the article by Cao and Kanafani (1997) which support our claim. These errors could be attributed to printing errors, but given the difficulties we have had in implementing the model, we find ourselves convinced that the model does not work.

Chapter 6

Using Heuristics to Solve DARP

The exact mathematical DARP-models described in chapter 4 were difficult to solve within a reasonable time frame. For that reason, this chapter will focus on new ways of handling DARP. To this end heuristics are sometimes better at providing practical and flexible solutions to problems which arise in the real world. This chapter describes a heuristic, which can solve DARP.

6.1 Motivation for Using Heuristics

One of the fundamental difficulties associated with DARP is the complexity of the problem: The cost of a certain decision (e.g. reassigning an aircraft to a flight) depends on other decisions made earlier. This dependency along with a vast solution space size make the problem difficult to solve.

A related problem is the speed with which DARP has to be solved to be of any use to flight planners. DARP arises when sudden disruptions occur in a planned flight schedule. In other words, there is most likely very little time to make recovery decisions. Flight controllers from British Airways claim that it must not take more than 2 minutes for the computer model to solve the DARP – otherwise they will not be able to use it.

Given these characteristics of DARP, it seems reasonable to take a new look at solving it. Flight planners are not necessarily interested in a proven optimal solution – they simply want a good solution, which they can start implementing as quickly as possible. Heuristics are often good at finding reasonably good solutions quickly, hence the motivation for attempting this.

6.2 Goals for Chapter 6

The goal of this chapter is to create a heuristic that solves DARP efficiently. The scope of this heuristic will be to include the same considerations that Cao and Kanafani (1997) did, although the **heuristic setting** will be a little different. The definition of DARP given in figure 2.1 does not specify a time horizon, i.e. how far into the future does the revised flight schedule extend. Therefore, a brief description follows of the setting in which the DAR-heuristic is relevant:

At a given point in time during a normal working day, one or more disruptions occur in the flight schedule. This point in time is called the **decision point**. Based on the original flight schedule and the disruptions that just occurred, the DAR-heuristic will calculate a revised flight schedule that spans the remaining part of that particular working day. This revised flight schedule will be made by delaying, cancelling and swapping aircraft while respecting the following flight control principles:

- Delays and cancellations should be minimized.
- Delays are preferable to cancellations.
- Only certain aircraft can be assigned each flight.

In section 2.1.1 many more principles are listed. However, for the sake of simplicity these will not be considered until chapter 7. Likewise, the problem instances along with the costs are kept simple in order to establish the merits of a heuristic approach before complicating it with further considerations.

6.3 Basic Heuristic Design

In this section the basic heuristic design is defined. This includes a definition of the network on which the heuristic is based, a neighborhood definition and the cost design principles.

6.3.1 Definition of Network

Figure 6.1: *Underlying network to use with a DAR-heuristic.*

Before the DAR-heuristic can be designed, a network is defined to base the heuristic on. This network is shown in figure 6.1 and is similar to the network introduced by Cao and Kanafani (1997). However, there are significant differences.

Cao and Kanafani (1997) assume that a time horizon of 3 aircraft to flight assignments in a link is sufficient. However, in **short-haul** disruptions management, there is a reasonably well defined working day. At the end of the day, aircraft will lay over at various airports in order to carry out the planned early morning flights. It is therefore of great importance that the necessary number of aircraft lay over at each airports. In other words, a time horizon spanning the remaining part of the working day is more practical if such layover restrictions have to be considered.

To cater for the layover restrictions, each link terminates in a sink node, thus indicating where the aircraft will lay over. This makes it possible to calculate how many aircraft end at each particular airport and use this information to evaluate the quality of the revised schedule. Layover restrictions (see the term **aircraft balance**) are not introduced until chapter 7, but the underlying network has to contain the possibility of introducing them later.

Another new feature in the underlying network is the cancellation aircraft node. By default, all cancellation aircraft nodes are assigned to the sink node. However, the cancellation aircraft can be assigned to all flights, thus cancelling them.

Finally, there are surplus aircraft. By default, these are connected by a forward arc to the sink node. Consequently, if the surplus aircraft is not used, it remains at the airport. Alternately, it can be assigned to a flight, thus keeping it from getting cancelled or delayed.

Choice of Solution Neighborhood

Given the network shown in figure 6.1, a neighborhood can be defined. Here it becomes apparent that there are certain similarities between DARP and the Quadratic Assignment Problem (QAP). QAP can be described as the problem of assigning a set of n facilities to a set of n locations with given distances between the locations and given flows between the facilities. The goal is to place the facilities on locations in such a way that the sum of the product between flows and distances is minimal. DARP could be formulated in a similar manner with flights corresponding to locations and aircraft to facilities. It is therefore not strange that the neighborhood used in the DAR-heuristics, shown later in this chapter, in principle is the one most commonly used in QAP.

The QAP neighborhood is most commonly defined by the set of permutations which can be obtained by exchanging two assignments (see Taillard (1991), Angel and Zissimopoulos (1998) and Stützle (1999)). The neighborhood $\mathcal{N}$ to a solution φ can in mathematical terms be expressed as follows:

$$\mathcal{N}(\varphi) = \{\varphi' \mid \varphi'_r = \varphi_s \wedge \varphi'_s = \varphi_r,\ r \neq s \wedge \varphi'_i = \varphi_i \ \forall\, i \neq r, s\} \quad (6.1)$$

The indexes i, r and s refer to individual assignments in the solution φ. Compared with the neighborhood used in QAP, the neighborhood used in DARP is slightly different. In DARP, aircraft cannot be assigned to flights in different airports. In the standard QAP, all assignments are legal by default. This difference reduces the size of the DARP solution space. In short, the neighborhood to a DARP solution consists of all those solutions that can be reached by making 1 feasible swap between 2 different aircraft.

The original QAP neighborhood definition gives rise to n^2 neighbors, where n is the number of locations/facilities. In DARP n is the number of flight/aircraft nodes. However, the neighborhood in DARP is a lot smaller because of restrictions on which flights the aircraft can be assigned to. This is very fortunate with respect to a fast implementation of the heuristic, because it is possible to explore more neighborhoods within the same time frame. This advantage has naturally been utilized in the implementation.

To illustrate this, consider a problem instances with 40 airports, 80 aircraft and approximately 300 flights. The average number of candidate flights for each of the approximately 300 aircraft nodes is only 8 (see description of candidate sets later). This results in a neighborhood size of roughly 2400 solutions. In a QAP with 300 locations/facilities the number of solutions in a neighborhood would be 90,000. Hence, the size of the DARP neighborhood is significantly reduced by utilizing restrictions on aircraft assignments.

Both the DARP and QAP neighborhood can be halved by utilizing that only $\frac{n^2}{2}$ *different* neighbors exist. This too has of course been included in the implementation.

Evaluating Solutions

The time it takes to explore a neighborhood also depends on how effectively the objective value is calculated for each solution. A reasonable way of doing this would be to track each aircraft through its link and calculate the contribution to the objective value. Cancellations would also be included this way, because cancellation aircraft are treated as a normal aircraft in this respect. In other words, each flight would be considered once, causing this method to result in a complexity of $O(n)$, where n is the number of flights. It is possible, however, to reduce this calculation significantly by only recalculating the cost of those links between which the reassignments occurred. This too is utilized in the DAR-heuristic implementation.

The advantage of this reduction is best illustrated by applying it to the example just used. By only recalculating the flights in the modified links, the solution evaluation only has to consider approximately 8 flights as opposed to 300 for each solution in the neighborhood.

Given the neighborhood size and the complexity of evaluating a single solution, the overall complexity of evaluating all solutions in a neighborhood is $O(n^3)$ where n is the number of flights. In practical terms, however, the complexity is significantly smaller. Hopefully, this means that the chosen design of the network and the consequent problem structure allows any heuristic to be fast.

6.3.2 Basic Parameters and Decision Variables

Following is a definition of the variables, parameters and sets used in all the heuristics that have been implemented to solve DARP.

A	$=$	set of nodes representing aircraft.
a	$=$	index for aircraft nodes.
F	$=$	set of nodes representing flights.
f	$=$	index for flight nodes.
F_a	$=$	subset of F consisting of candidate flights considered for aircraft a. If aircraft a is delayed beyond the time horizon, F_a is set to empty. In figure 6.1, F_a could reasonably consist of flights $\{4, 5, 6, 7\}$ for aircraft $a = 4$.
F_c	$=$	dynamic set containing flights f that have been cancelled.
r_f	$=$	the revenue of flight f.
d_{af}	$=$	the delay incurred if aircraft a is assigned to flight f. d_{af} is calculated as needed and takes all relevant previous assignments into consideration. d_{af} is measured in minutes
c_f	$=$	cancellation cost incurred if flight f is not flown. c_f is a function of r_f and β_f.
α_f	$=$	delay cost multiplier associated with each flight f.
β_f	$=$	cancellation cost multiplier associated with cancelling flight f.

The decision variable is:

$$x_{af} = \begin{cases} 1 & \text{if aircraft } a \text{ is assigned to flight } f \\ 0 & \text{otherwise} \end{cases}$$

To illustrate how the decision variable works, refer to figure 6.1. Here $x_{1,1} = 1$ because aircraft 1 is assigned to aircraft 1. Conversely, $x_{1,8} = 0$ because aircraft 1 is not assigned to aircraft 8.

6.3.3 Objective function

Given the network from figure 6.1 and the basic parameters and decision variables, the objective function can be defined.

$$\begin{aligned} Objective \quad = \quad & \sum_{a \in A} \sum_{f \in F} r_f \cdot x_{af} \\ & - \sum_{a \in A} \sum_{f \in F \setminus F_c} \alpha_f \cdot DF \cdot r_f \cdot d_{af} \cdot x_{af} \\ & - \sum_{a \in A} \sum_{f \in F_c} \beta_f \cdot r_f \cdot x_{af} \end{aligned} \tag{6.2}$$

The first component in the objective function is the total revenue. The second component is the total cost of delays. The constant DF is the percentage of the revenue r_f which is subtracted per minute delay of flight f (see section 6.4.2). The third component is the cost associated with cancellations.

In both the second and third component of the objective function, the revenue r_f is used directly to measure the cost because it seems a natural way to prioritize the flights. In principle this renders α and β unnecessary. However, a true revenue r_f is typically not calculated until several weeks after the flight has been flown – only then is the necessary data available. A revenue r_f calculated before the actual flight can only be based on forecasts, e.g. the number and types of passengers, the airports between which the flight is flown, the aircraft type carrying out the flight, etc. Disruptions to the flight schedule may render these forecasts obsolete, which is why α and β are relevant. They can quickly be assigned values which encourages a heuristic to find solutions that prioritize the flights according to the actual situation on hand.

6.4 Creating Problem Instances

Before describing the different heuristics, that have been implemented to to solve DARP, a general description of the problem instances is given along with a brief description of how they were created. The integrity of these problem instances is vital if the validity of the heuristic is to be tested properly.

In figure 6.2 an outline of the test instance generator is given. Simply put, a link is created for each aircraft by repeatedly selecting a destination airport at random until the time horizon is exceeded. This limits the length of a link to a maximum of 5 flights. It is important to notice that because airports are randomly selected, flights will not be concentrated around a few airports like they would be in a **hub-and-spoke system** – instead they will be more evenly distributed. Likewise, aircraft will typically not travel back and forth between the same two airports. These 2 factors mean that the generated flight schedules do not resemble real flight schedules from a practical point of view. However, it seems reasonable to assume that the complexity remains unchanged.

```
procedure Test Instance Generator
    repeat
        Select an aircraft
        Select an initial ready-time
        Select airport randomly where aircraft will start
        Decide if the aircraft is delayed
        repeat
            Select a destination airport randomly
            Update clock according to original ready-time
        until extending the link will exceed time horizon
    until the desired number of aircraft are in use
end
```

Figure 6.2: *Test instance generator outline.*

In table 6.1, the problem instances used to test the DAR-heuristics are listed. Each problem instance is created randomly based on a series of problem parameters. The influence of these parameters on the generated problem instance is explained in section 6.4.2.

6.4.1 Cost Design Principles

The cost design is very important if the solutions found by the heuristic have to appear realistic. Below is a description of how the costs are designed and the parameters which are used to weigh the various decision possibilities against each other.

Instance No.	Number of Airports	Number of Aircraft	Number of Flights
1	10	10	38
2	10	20	80
3	10	30	125
4	10	40	166
5	10	50	206
6	20	20	78
7	20	40	158
8	20	60	231
9	20	80	306
10	20	100	382
11	30	30	111
12	30	60	221
13	30	90	326
14	30	120	440
15	30	150	559
16	40	40	148
17	40	80	290
18	40	120	432
19	40	160	588
20	40	200	741
21	50	50	195
22	50	90	333
23	50	110	404
24	50	150	562
25	50	200	753

Table 6.1: *Dimensions of the problem instances.*

- Revenue
  - Revenues are randomly generated for each flight.
  - The revenue for a flight lies between \$3000 and \$6000.
- Delays
  - The difference between a flight's original departure time and its actual departure time constitutes the delay.
  - Flights never depart before their original departure time.
  - The *delay cost* is a function of the flight revenue and the delay measured in minutes (see objective function in equation 6.2).
  - A delay cost multiplier α is used to prioritize flights.
- Cancellations
  - A *cancellation cost* multiplier β is used to prioritize cancellations.

The idea is that flight planners adjust the values of α and β in order to achieve an acceptable flight schedule. For example, if β_f for flight f is set to a large number, then that particular flight is unlikely to be cancelled. If α_f for flight f likewise is increased, then the heuristic is unlikely to delay that flight significantly – if it can be avoided.

Overall, the presumption is that no computer model will be able to factor in all influences on the costs. The only way to arrive at an acceptable and feasible revised flight schedule is if the flight planners can interactively adjust the cost assignments, i.e. adjust the values of α and β. This way even minor influences can be considered by the computer model.

6.4.2 Parameter Values

Below follows a description of the parameters, which change throughout the the problem instances listed in table 6.1. In many ways these parameters are similar to those described in chapter 5, however, due to their significant impact on the size of the problem and solution space, they will briefly be described again.

Number of aircraft: The number of flights which are flown in a generated schedule is roughly proportional to the number of aircraft. This parameter is therefore an important factor in determining the problem size.

Number of airports: The number of airports has an indirect influence on the solution space size. The reason for this is that the relationship between the number of airports and aircraft determines the aircraft density at each airport. The aircraft density at each airport determines the size of the candidate sets (see below).

Candidate sets F_a: The candidate sets are very important with respect to the size of the solution space. The smaller these sets are, the smaller the solution space. In all the generated problem instances, an aircraft can be assigned to any flight leaving from its current airport, excluding those flights which presently are a part of that aircraft's link. This means that the size of the candidate sets is proportional to the number of airports. This feature is changed in chapter 7 to see what effect it has on the computing time. However, as illustrated earlier in this section, the solution neighborhood is reduced if the number of flights, that an aircraft can be assigned to, is reduced.

Number of surplus aircraft: Surplus aircraft are aircraft which are ready to fly but have not been assigned to any flight. Surplus aircraft are rare in real life aircraft schedules, so to ensure a certain realism in the problem instances, these are only used sparingly; each problem instance has 1 surplus aircraft.

Delay percentage: This parameter indicates the approximate number of aircraft that are delayed; 20% in all problem instances. This also means that on average 20% of all flights are delayed.

Number of flights: This is not actually a parameter. It is derived after a schedule has been created based on the above parameters.

Aside from the parameters just listed, there are parameters, which are not changed throughout the problem instances listed in table 6.1. These parameters are listed below along with their assigned values.

Time horizon: The length of a flight link is limited by the time horizon. The time horizon has been set to 600 minutes.

Flying time: Unless otherwise mentioned the **flying time** is defined as the time it takes an aircraft to travel the distance from the origin to the destination airport. It is assumed that all flights have the same flying time. This can easily be changed, but for the sake of simplicity the flying time has been set to 100 minutes.

Turn-around time: This parameter indicates the minimum time, that the aircraft must spend on the ground at a given airport before it is ready to fly again. For all aircraft at all airports, this time has been set to 10 minutes.

Delay factor DF: The delay factor determines how much revenue to deduct from the objective function given a certain delay. The delay factor is the percentage of the revenue that is removed per minute delay of a given flight; DF has been set to 1%.

α and β: The purpose of these constants were described in connection with the objective function (see equation 6.2). α has been set to

1 to indicate that all aircraft have the same priority with respect to delays. β has been set to 1.5 to indicate that a flight has to be delayed so much, that more than 1.5 times its revenue is lost due to delay costs, before the heuristic will cancel the aircraft. The possibility of changing α and β was incorporated in the implementation.

The values assigned to the parameters used in this heuristic are not realistic from a practical point of view. However, given the structure of the problem, most of these values do not influence the problem complexity. In other words, we believe that our problem instances are realistic enough to test the merits of our implemented heuristic and underlying network.

6.4.3 Example of a Problem Instance

An example of a flight schedule generated by the algorithm outlined in figure 6.2 is illustrated in figure 6.3.

The problem instance generator feeds the heuristic algorithm with the problem illustrated in figure 6.3. Below is a print-out of the same flight schedule.

```
FLIGHT LINKS
------------------------------------------------------------------------
Flight Scheduled Actual  Assignments
 link    take-   Ready-
          off     time
------------------------------------------------------------------------
   1      188     188    (3-3) ->   (14-14) ->   (8-9)
   2       97     357    (10-10) ->   (4-4) ->   (15-17)
   3      143     143    (2-2) ->   (12-12) ->   (6-9)
   4       69     369    (1-1) ->   (11-11) ->   (5-5) ->   (16-17)
   5        0     275    (13-17)
   6        0       0    (7-9)
   7        0       0    (9-9)
   8        0       0    (17-17)
------------------------------------------------------------------------
```

Figure 6.3: *Example of a problem instance with 2 airports, 4 normal aircraft and 2 surplus aircraft.*

Here link 2 and 4 are delayed. As a consequence of this delay, the revenue of link 2 and 4 suffers:

```
-----------------------------------
 Link   First     First     Link
  No   Aircraft  Aircraft  Revenue
       in Link     Type
-----------------------------------
   1      3       normal    10100
   2     10       normal   -17760
   3      2       normal    11200
   4      1       normal   -31200
   5     13       surplus       0
   6      7       surplus       0
   7      9       cancel        0
   8     17       cancel        0
-----------------------------------
 TOTAL OBJECTIVE VALUE =   -27660
-----------------------------------
```

This problem instance is very simple, but it serves to illustrate the network used and the solutions found by any heuristic used later. The solution shown below was derived using the ILS heuristic described in section 6.6, but it could probably easily have been derived using common sense. The optimized flight schedule derived by the model is shown below:

```
FLIGHT LINKS
-------------------------------------------------------------------------
Flight Scheduled Actual  Assignments
 link    take-   Ready-
          off     time
-------------------------------------------------------------------------
   1      188     188    (3-3) ->   (14-14) ->   (8-9)
   2        0     357    (10-17)
   3      143     143    (2-2) ->   (12-12) ->   (6-5) ->   (16-17)
   4        0     369    (1-9)
   5      179     275    (13-11) ->   (5-9)
   6       69       0    (7-1) ->   (11-10) ->   (4-4) ->   (15-17)
   7        0       0    (9-9)
   8        0       0    (17-17)
-------------------------------------------------------------------------
```

And the revenue achieved by each link:

```
-----------------------------------
 Link   First     First     Link
  No   Aircraft  Aircraft  Revenue
       in Link     Type
-----------------------------------
   1      3       normal    10100
   2     10       normal        0
   3      2       normal    12656
   4      1       normal        0
   5     13       surplus     184
   6      7       surplus    7398
   7      9       cancel        0
   8     17       cancel        0
-----------------------------------
 TOTAL OBJECTIVE VALUE =    30338
-----------------------------------
```

Both surplus aircraft have been used, because they incur smaller delays than the aircraft originally assigned. The result is a significantly improved flight schedule.

With this description of the problem instances, this chapter will move on to set up some basic principles for experimenting with heuristics before describing the heuristics applied to solve DARP.

6.5 Experimental studies

In Barr et al. (1995) guidelines are set up which encourage the scientific community to follow some basic principles when experimenting and reporting on different heuristics. It is argued that experimenting with heuristics should show the influence of different factors. A *factor* is any controlled variable in an experiment that influences the outcome or result. Such factors include:

Problem factors are various problem characteristics – such as dimensions, structure, distributions – that can affect the results of the heuristic.

Algorithm factors are for example strategies used in the heuristic method (e.g. search strategy) and parameters that control such strategies.

Test environment factors concerns the kind of *hardware, software, operating system* and for that matter also the *programmer* used for the experiments.

Barr et al. (1995) also cite the steps of the experimental process, which are:

1. Define the goals of the experiment
2. Choose the measures of performance and factors to explore
3. Design and execute the experiment
4. Analyze the data and draw conclusions
5. Report on the experimental results

Following the guidelines of Barr et al. (1995), experiments are designed that only adjust problem and algorithm factors. Factors concerning the test environment are not change throughout the experiments.

All the algorithms are programmed in *C++* using the *Gnu compiler* available on the *HP Unix* operating system *version 10.20.* The experiments are all performed on a *HP J7000, 440 MHz* workstation.

6.6 Iterated Local Search Heuristic

6.6.1 Motivation

The Iterated Local Search (ILS) heuristic is well described in Stützle (1999) who uses it on QAP. Because of the similarities in the nature of DARP and QAP, it is reasonable to assume that an ILS heuristic would work well on DARP. Presumably, several other heuristics would work well too, but as is the case with many heuristics, there is no well documented way to evaluate which method will work best.

6.6.2 Implementing the ILS Heuristic

The simplified program structure of the ILS heuristic is shown in figure 6.4.

As can be seen from the ILS algorithm, there are six main elements to implementing this heuristic. These will be described in the following:

```
procedure Iterated Local Search
    $x^0_{af}$ = GenerateInitialSolution
    $x_{af}$ = LocalSearch($x^0_{af}$)
    repeat
        $x'_{af}$ = Modify($x_{af}, history$)
        $x''_{af}$ = LocalSearch($x'_{af}$)
        $x_{af}$ = AcceptanceCriterion($x_{af}, x''_{af}, history$)
    until termination condition met
end Iterated Local Search
```

Figure 6.4: *Outline of an ILS Algorithm*

GenerateInitialSolution: The initial solution is simply the original flight schedule including the delays/cancellations.

LocalSearch: This procedure finds the local optimum in the neighborhood of the current solution/schedule.

Modify: This procedure makes some modifications in the current solution to enable the *LocalSearch* procedure to search other regions of the solution space.

AcceptanceCriterion: This procedure determines which solution to accept as a starting point for the next iteration.

history: This parameter will be explained at a later point.

termination condition: The termination condition will be time based. There may be other criteria which help to generate different revised schedules, but ultimately, the termination criteria will depend solely on the time allowed by flight planners.

Below follows a more in depth description of the procedures *LocalSearch, Modify* and *AcceptanceCriterion.*

Choice of *LocalSearch*

The local search procedure is initiated by a solution x_{af} in the form of a flight schedule. A best improvement strategy is chosen so that all of the neighbors to x_{af} are evaluated before the best solution x'_{af} among the neighbors to x_{af} is used as a starting point for the next iteration.

Choice of *Modify*

The purpose of this procedure is to ensure that the heuristic does not get trapped in a local optimum. The key to using local search heuristics effectively lies in how this is avoided (e.g. the stochastic element of simulated annealing or the taboo list in taboo search). In very general terms, any heuristic making use of local search will, once it has reached a local optimum, select a new solution from which to conduct a new local search. This selection is either done by randomly selecting a new solution or by modifying an existing one.

In this case, *Modify* changes the current solution x_{af} by making a number of random swaps. The local search is then applied to the modified solution x'_{af}: Because of the random swaps just made, it is possible for the local search to find new and better local optima. If not, a number of new random swaps can be made in x_{af}. This continues until some termination condition is met. In short, *Modify* chooses which swaps to make and how many.

Choice of *AcceptanceCriterion*

The acceptance criterion is very simple. A function *Better(x_{af}, x''_{af})* is used and is defined as follows:

$$x_{af} := \mathit{Better}(x_{af}, x''_{af}) = \begin{cases} x''_{af} & \text{if } f(x''_{af}) > f(x_{af}) \\ x_{af} & \text{otherwise} \end{cases} \tag{6.3}$$

$f(x_{af})$ is the objective function value of the solution x_{af}. This is according to Mladenović and Hansen (1997) the most common way to define the acceptance criterion. Allowing moves to worse solutions may yield better heuristic performance as the case is with most metaheuristics, but this will not be explored here.

Final ILS Outline

To control the number of swaps made by *Modify*, a Variable Neighborhood Search (VNS) is embedded in the ILS framework. VNS is well described in Mladenović and Hansen (1997) and provides a method of selecting how many random swaps are made in the current solution.

When a new local optimum is reached, it seems reasonable to search for better solutions in the vicinity of this optimum before exploring more distant solutions. In the ILS/VNS framework this means that the number of random swaps made in the current solution increases every time a better solution has *not* been found using *LocalSearch*. This principle is formalized in in figure 6.5 which illustrates the ILS procedure with VNS embedded. Steps *a, b* and *c* correspond to *Modify*, *LocalSearch*, and *AcceptanceCriterion* respectively.

```
**Initialization** *Variable Neighborhood Search*
(1)  Select a set of neighborhood structures $\mathcal{N}_k,\ k = \{1 \ldots k_{max}\}$
(2)  Select an initial locally optimal solution $x_{af}$
**Main Procedure** *Variable Neighborhood Search*
(1)  Set $k := 1$
(2)  **repeat**
       a.  generate at random a solution $x'_{af}$ in the $k^{th}$ neighborhood
           of $x_{af}$
       b.  conduct local search from $x'_{af}$; denote locally optimal
           solution $x''_{af}$
       c.  if $x''_{af}$ is better than $x_{af}$:
             continue search from $\mathcal{N}_1$, $(k := 1)$
           else:
             set $k := k + 1$
     **until** $k = k_{max}$ or other termination criterion is met.
```

Figure 6.5: *Outline of the ILS procedure incorporating VNS.*

The outline in figure 6.5 contains two elements that require a brief explanation. Firstly, the variable k needs to be explained. This variable corresponds to the *history* variable used in figure 6.4. Simply put, k keeps track of how many random swaps to perform in the *Modify* procedure.

Secondly, $\mathcal{N}_k$ needs to be defined. The k^{th} neighborhood structure to a solution x_{af} is defined as:

> *The neighborhood to a solution x_{af} where k random swaps have been made.*

The modification of the best solution so far is what makes the ILS heuristic *iterative*. By searching neighborhoods of solutions generated from the current solution, it is likely that positive characteristics are retained. Often successive generated neighborhoods ($\mathcal{N}_k$) will be nested, but if k_{max} is large enough, the heuristic is likely to eventually escape the local optimum.

6.6.3 Experimental goals

To test the ILS heuristic, a set of experiments have been designed with the following goals:

- Show the influence of k_{max} on the quality of the solutions produced and possibly determine which k_{max} give the best solutions.
- Analyze the degree of improvement as a function of the running time.
- Design experiments that run for a duration of 24 hours in order to possibly find better solutions than have previously been found.
- Show which values of k actually yield improvements during the search.
- Show if there is a correlation between the size of the problem (i.e. the number of flights in the schedule) and the best k_{max}.

The resulting data from the experiments is documented in 2 places: In the case of the 3-minute test runs, the resulting data can be found in appendix C.1; the 24-hour test runs can be found in appendix C.2.

Results of the Analysis of the Parameter k_{max}

The histogram shown in figure 6.6 illustrates the results from the 3-minute test runs. Here the number of times the best solution was found for a given k_{max} are counted. For example, in 6 of the problem instances, the best solution found in the 3-minute test runs was found with $k_{max} = 1$. Notice that the total number of observations in figure 6.6 is greater than 25 (the number of problem instances). The reason for this is that identical best solutions to certain problem instances were found for several different values of k_{max}.

It is reasonably clear, that there is no value of k_{max}, which repeatedly finds the best solution. However, there does seem to be a slight tendency for smaller values of k_{max} to result in good solutions. In the histogram in figure 6.7 a different principle was applied: If identical best solutions to a

Figure 6.6: *The number of times the best solution was found for a given* k_{max}.

certain problem instance were found for several values of k_{max}, only the smallest k_{max} is counted.

Figure 6.7: *The number of times the best solution was found for a given* k_{max} *when only the smallest* k_{max} *for each problem instance is counted.*

The histogram in figure 6.7 still fails to determine the best value of k_{max}. However, smaller values of k_{max} may on average produce better results. To analyze the *average* quality of the solutions found by the heuristic, another analysis is made. The objective values found for each value of k_{max} are normed relative to the best objective value ever found for that test instance. The normed values for each value of k_{max} are then accumulated over the

25 test instances and the result of this analysis is shown in figure 6.8.

Figure 6.8: *Graph of the accumulated normed values as a function of* k_{max}.

Due to the very narrow interval of the abscissa axis the analysis only shows a weak correlation between the quality produced and the various values of k_{max}. It seems the quality decreases with an increasing value of k_{max}, i.e. the smallest average deviation from the best objective value is found for $k_{max} = 1$.

In table 6.2 the best results found for each problem instance by the 3-minute ILS test runs are listed. For the sake of comparison, these results are compared to the best solutions ever found to the problem instances. With an average gap of only 0.68% between the results, the ILS heuristic is remarkably close.

Run-time Analysis

In the 3-minute test runs, the best objective value found so far is logged at specified points in time. This is done in order to see the degree of improvement as a function of the running-time.

In figure 6.9 the result of the run-time analysis is illustrated. For each k_{max} the total number of seconds spent on finding a solution for each problem instance is shown. It is only the time it took to find a solution within 5% of the best objective value ever found that is counted. As it can be seen the average time it takes ILS to find such a solution for each problem instance increases with increasing k_{max}. Based on this alone, a small k_{max} seems most effective in the 3-minute runs.

Instance No.	Number of Flights	Best Result	Standard ILS	Gap (in %)
1	38	72697	72697	0.00
2	80	264051	264051	0.00
3	125	402593	398337	1.06
4	166	614664	610489	0.68
5	206	690592	689158	0.21
6	78	201030	201030	0.00
7	158	508363	500279	1.59
8	231	814305	811805	0.31
9	306	1096463	1091002	0.50
10	382	1292384	1277930	1.12
11	111	376766	376766	0.00
12	221	782862	778197	0.60
13	326	1110793	1106776	0.36
14	440	1569761	1564826	0.31
15	559	1763103	1752311	0.61
16	148	437393	423691	3.13
17	290	963745	962878	0.09
18	432	1521388	1521328	0.00
19	588	1954775	1940366	0.74
20	741	2407012	2350704	2.34
21	195	612507	610775	0.28
22	333	1127078	1124859	0.20
23	404	1473042	1458897	0.96
24	562	1773247	1761924	0.64
25	753	2595359	2560917	1.33
Avg. gap between best solution and ILS (3 min):				0.68

Table 6.2: *Overview of the best results found by ILS when given a running time of 3 minutes.*

Results of the 24-hour Experiments

The 24-hour test runs were conducted in order to examine how well the ILS heuristic performed if it were given an increased run-time. A $k_{max} = 8$ was chosen to ensure the heuristic a reasonable possibility of escaping local optima. The results of these test runs can be seen in appendix C.2, however, they are also summarized in table 6.3.

Interestingly, table 6.3 shows that the ILS heuristic does not find better solutions given a longer run-time. Reasons for this will be explored in section 6.8.

Figure 6.9: *The average number of seconds used by the ILS heuristic to find a solution for each problem instance. By solution is meant one which has an objective value of more than 95% of the best objective value ever found. This is done for each* k_{max}.

Analysis of k-values

Besides examining which values of k_{max} yield good results, it would be interesting to see which values of k *actually* yield improvements. The variable k is described in figure 6.5 and takes on all integer values between 1 and k_{max}. The observations of k that yield improvements are accumulated during the 24-hour test runs for all the test instances; the resulting histogram is shown in figure 6.10.

It can be seen that for $k = 1$, more than 80% of the improvements are made. A total of only 16 improvements were made for $k \geq 4$, which means that in at least 9 problem instances, improvements were only made for $k \leq 3$. Furthermore, no improvements were found at all for $k = 8$, so a $k_{max} > 7$ should do nothing to improve the solutions generated by the ILS heuristic. Figure 6.8 illustrates something else; on average, better solutions are found with $k_{max} = 8$ than with $k_{max} = 7$. This does not necessarily indicate

Instance No.	Number of Flights	Best Result	Standard ILS	Gap (in %)
1	38	72697	72697	0.00
2	80	264051	263450	0.23
3	125	402593	386736	3.94
4	166	614664	607477	1.17
5	206	690592	688226	0.34
6	78	201030	201030	0.00
7	158	508363	500279	1.59
8	231	814305	803270	1.36
9	306	1096463	1095148	0.12
10	382	1292384	1276954	1.19
11	111	376766	376766	0.00
12	221	782862	778559	0.55
13	326	1110793	1100193	0.95
14	440	1569761	1569378	0.02
15	559	1763103	1763103	0.00
16	148	437393	423691	3.13
17	290	963745	963691	0.01
18	432	1521388	1506861	0.95
19	588	1954775	1954775	0.00
20	741	2407012	2407012	0.00
21	195	612507	610155	0.38
22	333	1127078	1126976	0.01
23	404	1473042	1472573	0.03
24	562	1773247	1761621	0.66
25	753	2595359	2595359	0.00
Avg. gap between best solution and ILS (24 hours):				0.67

Table 6.3: *Overview of the results found by ILS when given a running time of 24 hours.*

a contradiction because of the randomized selection of which flights and aircraft to swap. Put simply, ILS may simply have been more fortunate with its swaps when $k_{max} = 8$.

Correlation Between Problem Size and k_{max}

All the attempts to determine the values of k_{max} that yield the best results are somewhat inconclusive. There only seems to be a slight tendency for smaller values of k_{max} to work better. However, it is possible that there is a correlation between the values of k_{max}, which yielded the best solutions, and the problem instance size. In figure 6.11 k_{max} is plotted as a function of the number of flights. However, given our present test instances, there is no significant correlation between the two parameters.

Figure 6.10: *Frequency of the k which yielded the actual improvements in the solution during the search.*

Figure 6.11: *Plot of the values of the k_{max} as a function of the number of flights. Only the k_{max} which yielded the best solutions were included.*

6.6.4 Conclusions on the ILS Heuristic

The experiments with the ILS heuristic are not able to establish a single most efficient value of k_{max}. On average the best solutions are produced when $k_{max} = 1$, but the quality only decreases slightly when k_{max} increases.

The run-time analysis showed that the average amount of time spent by ILS on finding solutions close to the best solutions ever found increased as k_{max} increased.

Experiments that run for a duration of 24 hours do not yield better solutions on average. In other words, increased running time does not improve the solutions produced by the ILS heuristic. This may indicate that the ILS heuristic is unable to escape local optima. It may also indicate that the ILS finds solutions quickly that are close to being globally optimal.

The analysis of the k values, which actually yielded improvements, only confirmed that k_{max} should be set to a relatively small value.

There is no correlation between the size of each problem instance and the associated k_{max}, which yielded the best solutions

6.7 The Revised Iterated Local Search Heuristic

6.7.1 Motivation

It is reasonable to assume that the quality of the solutions produced by ILS depend on the initial solution given. In this section, the ILS heuristic is modified by conducting a Steepest Ascent Local Search first. The solution reached this way is used as the starting point for the ILS heuristic. Possibly, the Revised Iterated Local Search (RILS) is more effective and the optimal values of its parameters easier to identify.

6.7.2 Implementing the RILS Heuristic

The modifications to the structure of the ILS heuristic only concerns the initials steps. The general outline can be seen in figure 6.12.

An outline of the procedure *SteepestAscentLocalSearch* used in the Revised ILS is described later in figure 6.16. The other functions are all identical with the ILS heuristic outline shown in figure 6.4.

6.7.3 Experimental goals

The goals of experiments with RILS are similar to those of the experiments with ILS. However, aside from the influence of k_{max}, the RILS heuristic is

```
**procedure** *Revised Iterated Local Search*
    $x^0_{af}$ = GenerateInitialSolution
    $x_{af}$ = SteepestAscentLocalSearch($x^0_{af}$)
    **repeat**
        $x'_{af}$ = Modify($x_{af}, history$)
        $x''_{af}$ = LocalSearch($x'_{af}$)
        $x_{af}$ = AcceptanceCriterion($x_{af}, x''_{af}, history$)
    **until** termination condition met
**end** *Revised Iterated Local Search*
```

Figure 6.12: *Outline of the Revised ILS Algorithm*

assumed to behave like ILS. For this reason there are only 2 experimental goals:

- Show the influence of k_{max} on the quality of the solutions.
- Compare the quality of solutions achieved by RILS and ILS.

The resulting data from the experiments are documented in appendix C.3.

Results of the Analysis of the Parameter k_{max}

With regard to the optimal value of k_{max}, there seems to be a clearer tendency in RILS. In figure 6.13 a histogram is shown. In this histogram, the number of times the best solution is found for a given k_{max} is counted. As in figure 6.7 only the smallest k_{max}, for which the best solution was found, are counted.

In figure 6.13 it can be seen that over 50% of the best results were found with $k_{max} = 1$.

As with the Standard ILS, an analysis of the average quality of the solutions produced is conducted and the result of this analysis is shown in figure 6.14.

Here the interval on the abscissa axis is even smaller than for the graph in figure 6.8. It seems like the Revised ILS produces solutions of the same quality regardless of the value of k_{max}. This indicates that the parameter for this type of problem is superfluous.

Figure 6.13: *The number of times the best solution was found for a given k_{max} when only the smallest k_{max} for each problem instance are counted.*

Figure 6.14: *Graph of the accumulated normed values as a function of k_{max}.*

Results of the RILS Heuristic

The results of the experiments with a time duration of 3 minutes are summarized in table 6.4. As can be seen, the gap between the solutions achieved by RILS and the best solution is slightly better than that achieved by ILS.

6.7.4 Conclusions on the RILS Heuristic

The influence of k_{max} is virtually non-existent. Figure 6.14 clearly illustrates that RILS produces solutions of similar quality regardless of which

Instance No.	Number of Flights	Best Result	RILS	Gap (in %)
1	38	72697	72697	0.00
2	80	264051	264051	0.00
3	125	402593	397670	1.22
4	166	614664	614664	0.00
5	206	690592	690592	0.00
6	78	201030	196291	2.36
7	158	508363	507352	0.20
8	231	814305	808205	0.75
9	306	1096463	1091895	0.42
10	382	1292384	1292384	0.00
11	111	376766	376766	0.00
12	221	782862	776675	0.79
13	326	1110793	1105155	0.51
14	440	1569761	1556952	0.82
15	559	1763103	1748860	0.81
16	148	437393	423560	3.16
17	290	963745	963745	0.00
18	432	1521388	1512786	0.56
19	588	1954775	1934650	1.03
20	741	2407012	2401028	0.25
21	195	612507	608430	0.67
22	333	1127078	1127078	0.00
23	404	1473042	1473042	0.00
24	562	1773247	1769231	0.23
25	753	2595359	2579740	0.60
Average gap between best solution and RILS (3 min):				0.57

Table 6.4: *Difference between Revised ILS and best result obtained by any of the heuristics implemented.*

k_{max} is chosen. This renders k_{max} superfluous from a practical point of view.

The experiments with the RILS heuristic are not able to show its superiority over the ILS heuristic. A different initial solution is therefore not a worthwhile modification to the ILS heuristic.

6.8 Steepest Ascent Local Search Heuristic

6.8.1 Motivation

Both ILS algorithms illustrated that reasonably good solutions were found quickly – typically within the first 10 seconds. Moreover, when $k_{max} = 1$, the solutions found by the heuristics were slightly better on average than those solutions found for all other values of k_{max}. The interesting thing about this fact is that an ILS algorithm with $k_{max} = 1$ is very similar to a Steepest Ascent Local Search (SALS) algorithm (see figure 6.16). The only difference is the function *Modify* shown in figure 6.4; when $k_{max} = 1$ the modification made by *Modify* is very small. If a modification is made that worsens the solution significantly, then the following *LocalSearch* function will most likely undo that modification. Hence, there is very little difference between ILS and SALS for $k_{max} = 1$ – except that SALS presumably is faster.

Another interesting observation is that the 24-hour test runs did not improve the solutions found by either ILS heuristic significantly. One reason for this could be that the solution space has a shape like that illustrated in figure 6.15.

Figure 6.15: *Geometric fitness landscape as a function of all combinations of values assigned to the decision variables*

If the fitness landscape of the problem instances looks like that illustrated in figure 6.15, then increased computational time is not going to improve the results significantly. This is because significantly better solutions do not exist once the "hill has been climbed".

A SALS algorithm "climbs the hill" very quickly, though it easily risks getting trapped in a local optimum like the one also illustrated in figure 6.15. However, it might find reasonably good solutions very *quickly*, which is the motivation for implementing a SALS algorithm.

6.8.2 Implementing the SALS Heuristic

The simplified program structure of the SALS algorithm is shown in figure 6.16. There are 3 main elements and these are:

```
procedure Steepest Ascent Local Search (SALS)
    $x_{af}$ = GenerateInitialSolution
    repeat
        $x'_{af}$ = LocalSearch($x_{af}$)
        $x_{af}$ = AcceptanceCriterion($x_{af}, x'_{af}$)
    until a better solution cannot be found
end SALS
```

Figure 6.16: *Outline of a SALS procedure*

GenerateInitialSolution: The initial solution is the original flight schedule including the delays/cancellations – as in the ILS algorithm.

LocalSearch: This procedure finds the best local optimum in the neighborhood of the current solution/schedule.

AcceptanceCriterion: This procedure determines which solution to accept as a starting point for the next local search. Just like in the ILS algorithm, the acceptance criterion in SALS uses a function $Better(x_{af}, x'_{af})$ shown in equation 6.4.

$$x_{af} := Better(x_{af}, x'_{af}) = \begin{cases} x'_{af} & \text{if } f(x'_{af}) > f(x_{af}) \\ x_{af} & \text{otherwise} \end{cases} \qquad (6.4)$$

6.8.3 Experimental Goals

There are 2 experimental goals:

- Investigate the quality of the solutions generated by SALS when compared to the solutions generated by the 2 ILS heuristics.
- Investigate the run time of SALS.

Instance No.	Number of Flights	Best Result	SALS	Gap (in %)	Time (in secs.)
1	38	72697	59008	18.83	0.01
2	80	264051	255213	3.35	0.09
3	125	402593	375699	6.68	0.12
4	166	614664	594911	3.21	0.50
5	206	690592	675224	2.23	0.90
6	78	201030	196291	2.36	0.03
7	158	508363	507043	0.26	0.26
8	231	814305	803711	1.30	0.63
9	306	1096463	1086028	0.95	1.29
10	382	1292384	1268061	1.88	3.16
11	111	376766	369218	2.00	0.02
12	221	782862	776675	0.79	0.34
13	326	1110793	1100121	0.96	1.12
14	440	1569761	1538055	2.02	2.62
15	559	1763103	1683127	4.54	6.77
16	148	437393	417642	4.52	0.07
17	290	963745	958332	0.56	0.57
18	432	1521328	1506960	0.94	1.70
19	588	1954775	1917987	1.88	5.19
20	741	2407012	2336639	2.92	12.15
21	195	612507	602827	1.58	0.15
22	333	1127078	1104572	2.00	0.63
23	404	1473042	1449719	1.58	0.85
24	562	1773247	1721653	2.91	4.05
25	753	2595359	2547788	1.83	9.42
Average gap between best solution and SALS:				2.22	(secs.) 2.11

Table 6.5: *Overview of the SALS heuristic results*

Solution Quality of SALS

In table 6.5 the results achieved by the SALS algorithm are listed.

Firstly, there appears to be a something different about problem instance 1. It is the smallest problem instance and presumably one swap can make a very large difference. Thus, SALS probably ended in a local optimum early and therefore did not make a few more swaps that would have changed the result significantly. Because the result achieved problem instance 1 is so different from all other results, it has *not* been included in any of the statistics.

In relation to the first experimental goal, SALS finds solutions that are almost as good as the best solutions found by either ILS heuristic. The average gap percentage between the best solution ever found and the best solution found by the SALS and ILS heuristics are listed in table 6.6.

	SALS	Standard ILS	Revised ILS
Average run time	2.11 secs.	3 min.	3 min.
Average gap	2.22%	0.68%	0.57%

Table 6.6: *Average gap between the best solution ever found for each problem instance and the* ***best*** *solution found by SALS and ILS respectively.*

It is clear that both ILS heuristics find better solutions than SALS. However, they are only better by a very small margin. The ILS gap percentages listed in table 6.6 are also based on the best results found. These results were only found for some values of k_{max}; as mentioned, it was not possible to discern any correlation between the problem instance characteristics and the k_{max} which yielded the best solutions. In other words, there is no way of predicting which k_{max} finds the optimal solution. It would therefore be more accurate to compare the SALS results with the average solutions found by ILS for each problem instance. This is done in table 6.7.

	SALS	Standard ILS	Revised ILS
Average run time	2.11 secs.	3 min.	3 min.
Average gap	2.22%	1.75.%	1.27%

Table 6.7: *Average gap between the best solution ever found for each problem instance and the* ***average*** *solution found by both ILS heuristics.*

The difference in the quality of the solutions found by SALS and ILS are very small – even smaller if the SALS results are compared with the average ILS results. However, it is also important to analyze if they behave similarly, i.e. how do the result found by SALS compare with those found by ILS for each problem instances?

As can be seen in figure 6.17, all three heuristics seem to behave similarly in all the problem instances – irrespective of the problem size. This fact also justifies the gap comparison in figure 6.7 and 6.6.

It should be noted that the best solutions found to the problem instances listed in table 6.1 improved the revenue of the original flight schedule by 59.0% on average. In other words, whether it is ILS or SALS that is used, the revenue improvement is still approximately 57% on average.

Figure 6.17: *The gap between three heuristic approaches and the best solutions ever found as a function of the problem size.*

Run-time Analysis

In relation to the second experimental goal, SALS is clearly very fast. ILS was run for 3 minutes for all problem instances, whereas SALS stops when it has reached a local optimum. In table 6.5, it takes 2.11 seconds on average to find a local optimum using SALS and never more than 13 seconds. It is therefore obvious that SALS is a lot faster than ILS.

6.8.4 Conclusions on SALS

All other things equal, the ILS heuristics produce better results than SALS. However, the speed with which SALS finds solutions makes it very attractive. With respect to practical implementation at an airline, SALS would certainly be the most well-suited algorithm to solve DARP.

6.9 Repeated Steepest Ascent Local Search Heuristic

6.9.1 Motivation

Given the success of SALS, it seemed reasonable to try SALS with different initial solutions – a Repeated Steepest Ascent Local Search Heuristic (RSALS). This would most likely allow SALS to find other local optima, possibly even ones which were better than those previously found by ILS.

6.9.2 Implementing the RSALS Heuristic

The simplified program structure of the RSALS algorithm is shown in figure 6.18.

```
procedure Repeated Steepest Ascent Local Search (RSALS)
    $x^0_{af}$ = GenerateInitialSolution
    repeat
        $x_{af}$ = Modify($x^0_{af}$)
        repeat
            $x'_{af}$ = LocalSearch($x_{af}$)
            $x_{af}$ = AcceptanceCriterion1($x_{af}, x'_{af}$)
        until A better solution cannot be found
        $x^*_{af}$ = AcceptanceCriterion2($x_{af}, x^*_{af}$)
    until Stopping criterion is met
end RSALS
```

Figure 6.18: *Outline of an RSALS procedure*

As can be seen from the RSALS algorithm illustrated in figure 6.18, there are 5 main elements. These are:

GenerateInitialSolution: The initial solution is the original flight schedule including the delays/cancellations – as in the ILS algorithm.

Modify: This procedure modifies the original flight schedule by making up to 35 random swaps in the original solution. This assures a different starting point for each SALS.

LocalSearch: This procedure finds the best local optimum in the neighborhood of the current solution/schedule.

AcceptanceCriterion1: Same as AcceptanceCriterion described for SALS earlier.

AcceptanceCriterion2: This function ensures that the best solution found overall is stored. This is again done through a function like that shown in equation 6.4.

RSALS made 2000 iterations for each problem instance. One iteration refers to one SALS algorithm like that described in figure 6.16.

6.9.3 Experimental Goals

There are 2 experimental goals:

- To see if it is possible to find better solutions to the 25 problem instances in table 6.1 than have previously been found.
- Gather data for a fitness landscape analysis.

Results of the RSALS Tests

In table 6.8 the results achieved by the RSALS algorithm are listed.

In several cases RSALS finds a better solution to a problem instance than had been found before. On average the solutions found were also very good with an average gap of only 0.18% between the best solutions ever found and the solutions found by RSALS. RSALS spent anywhere between 30 minutes and 6 hours on making 2000 iterations in each problem instance, so for practical purposes it is not very well suited for flight operations decision planning.

6.9.4 Conclusions on RSALS

On average RSALS found the best solutions compared with any of the other heuristics. However, due the computational time needed, RSALS is not practical. With respect to the fitness landscape analysis, this is covered separately in the following section.

Instance No.	Number of Flights	Best Solution	RSALS ILS	Gap (in %)
1	38	72697	72697	0.00
2	80	264051	264051	0.00
3	125	402593	402593	0.00
4	166	614664	614116	0.09
5	206	690592	689971	0.09
6	78	201030	201030	0.00
7	158	508363	508363	0.00
8	231	814305	814305	0.00
9	306	1096463	1096463	0.00
10	382	1292384	1287860	0.35
11	111	376766	375777	0.26
12	221	782862	782262	0.00
13	326	1110793	1110793	0.00
14	440	1569761	1569761	0.00
15	559	1763103	1761276	0.10
16	148	437393	437393	0.00
17	290	963745	963534	0.02
18	432	1521388	1512388	0.59
19	588	1954775	1949106	0.29
20	741	2407012	2382363	1.02
21	195	612507	612507	0.00
22	333	1127078	1122299	0.42
23	404	1473042	1471853	0.08
24	562	1773247	1773247	0.00
25	753	2595359	2569895	0.98
Average gap between best solution and RSALS:				0.18

Table 6.8: *Overview of the RSALS heuristic results.*

6.10 Analysis of the Search Space

The speed with which SALS found a good solution calls for a further analysis of the nature of the solution space. It is unusual that a relatively simple SALS algorithm finds very good solutions compared to customized heuristics that run for 24 hours. However, the results show this.

In figure 6.19 fitness landscapes have been made for 5 problem instances (refer to table 6.1). The data for these fitness landscapes were generated using the RSALS heuristic described in the previous section.

Each fitness landscape uses the best solution ever found for a particular problem instance as a reference point. Technically this reference point is the point (0,0) in each figure. All the other points represent other local optima found by RSALS. In all, there are 2000 such local optima in each fitness landscape. For each of these local optima the distance to the reference optimum is calculated. Here distance is a simple count of the number of assignments in the local optima, which are different from the assignments in the reference optimum. This value is plotted against the numerical difference in the corresponding objective values. The result is a scatter diagram, which may show some correlation between the objective values and the distances.

In all the fitness landscapes in figure 6.19 there is a remarkably clear correlation: The further the distance to the best solution ever found, the worse the objective values get. To illustrate this, a linear fit has been made in all the fitness landscapes. This fit does not explain so much of the observed variation. However, the tendency is clear and statistically significant. In other words, the fitness landscapes give a strong indication that the solution space may indeed look like that illustrated in figure 6.15.

It can be noticed in each fitness landscape that local optima seem to be grouped a certain minimum distance from the reference point. In figure 6.19(a) there are no local optima with a distance of less than 20 to the reference optimum. Although no data exists to prove it, it is reasonable to assume that the common distance between *all* local optima is at least 20 in figure 6.19(a). This is also supported by the fact that the minimum distance to the reference optimum increases as the problem size increases: It seems reasonable that the common distance between local optima in large problems is greater than that of small problems.

6.11 Conclusion on Chapter 6

It is clear that all the heuristics effectively are able to solve DARP. More specifically, two different ILS heuristics on average found better solutions than SALS, though by a very small margin. In fact, the margin is so small that from a practical point of view, it does not make any significant difference. SALS was extremely fast compared to the ILS algorithms, which makes it very suitable for disruption management. The speed with which the 25 problem instances were optimized would indicate that a series of further considerations could be added while retaining acceptable computational time.

A fitness landscape analysis clearly justified using a SALS algorithm; provided the landscapes look like that shown in figure 6.19, SALS can be expected to do well. However, when further considerations are added, the fitness landscape may change rendering SALS less effective. Further analysis would be needed to confirm this. In chapter 7 a number of such considerations are discussed.

(a) Fitness landscape for problem instance 4

(b) Fitness landscape for problem instance 9

(c) Fitness landscape for problem instance 14

(d) Fitness landscape for problem instance 19

(e) Fitness landscape for problem instance 24

Figure 6.19: *Fitness landscapes for 5 of the problem instances listed in table 6.1.*

Chapter 7

Further Development of the DAR-heuristic

The previous chapter described a successful implementation of a DAR-heuristic. However, a lot of considerations were not included explicitly. It seemed reasonable to test the merits of the heuristic approach with only the basic elements of DARP, before adding more detailed considerations. Such considerations include:

- Aircraft balance
- Flight schedule structure
- Swap costs
- Maintenance
- Airport curfews
- Passenger flow
- Multiple fleets
- Ferrying of aircraft

This chapter will describe how these considerations can be included in a DAR-heuristic. It should be mentioned that none of these considerations have been implemented: Doing so is in some cases relatively easy, but to verify the effect of adding more considerations, more realistic problem instances are needed. These are very difficult and time consuming to create and are in themselves not very interesting with regard to this thesis. However, a description of how the considerations could be added *is* interesting.

7.1 Aircraft Balance

A certain number of aircraft are supposed to end at each particular airport according to the original flight schedule. If the originally intended number of aircraft end at each airport, then there is said to be **aircraft balance**. Aircraft balance is necessary in order to service out-bound flights the next day and if there is an imbalance, some of these will have to be cancelled.

None of the DAR-heuristics implemented in chapter 6 explicitly considered aircraft balance. it is however, possible to include such considerations, although it should be noted that there is a direct trade-off between aircraft balance and flight cancellations.

Imbalance only arises when flights are cancelled. A simple swap between two normal aircraft will never cause an imbalance because the same number of aircraft end at each airport. However, a single swap between a normal aircraft and a cancellation aircraft will most often cause an imbalance. The trade-off between cancellations and imbalance is illustrated in figure 7.1 and 7.2.

Figure 7.1: *Basic flight schedule.*

Figure 7.1 illustrates a basic original schedule. There are two links in this schedule – each being initiated by aircraft 1 and 5 respectively. In case

Figure 7.2: *Basic flight schedule with 2 different swaps.*

A in figure 7.2 a swap has been made between aircraft 1 and 2. If each link is examined it should be clear that one aircraft ends at each airport – in accordance with the original solution. However, in case B a swap is made between aircraft 2 and cancellation aircraft 4. Now both links end in airport 1, i.e. an imbalance has occurred.

Therefore, if the possibility of imbalances is unacceptable, then the heuristic is limited to a certain type of cancellation. This kind of cancellation is illustrated in figure 7.3.

This figure illustrates that cancellations can be made, which do not cause imbalances to occur. Aircraft 1 has been swapped with cancellation aircraft 4, however, the same number of aircraft still end at airport 1. In short, these cancellations are possible if the cancelled flight is part of a link that ends at the airport from which the cancelled flight should have departed, e.g. the link initiated by aircraft 1 ends at airport 1 and flight 1 likewise departs from airport 1. Changing the current DAR-heuristic design to only make this kind of cancellation is possible and relatively uncomplicated.

Some types of cancellation are not possible if the above method is used. Again this is best explained in an illustration as in figure 7.4. Here a set of cancellations is made: Flights 2 and 6 have been cancelled, but one aircraft still ends at each airport, thus maintaining the aircraft balance.

This type of cancellation cannot be made given the current neighborhood definition (see equation 6.1) and the restriction that no swaps may cause

Figure 7.3: *Example of a cancellation in the basic flight schedule which does not cause an imbalance.*

Figure 7.4: *Example of a set of cancellations that does not cause an imbalance.*

an imbalance to arise. The reason is that a neighbor to a solution is found by making *one* swap – and making just one of the two swaps in figure 7.4 will cause an imbalance. This limitation on possible cancellations may not be a serious problem. This depends on the structure of the flight plan as will be discussed in section 7.2.

It should be mentioned that the set of swaps in figure 7.4 could be made if the neighborhood definition was changed to include all those solutions that can be reached by making *two* swaps. The disadvantage of this approach is a significant increase in computational time; in the worst case, the number of neighbors increase from n^2 to n^4 where n is the number of flights (see section 6.3.1).

7.1.1 Aircraft Balance in the Current DAR-heuristic

Before altering the current DAR-heuristic, it would be interesting to evaluate how it actually performs with regard to aircraft balance. For this purpose the RSALS-heuristic described in section 6.9 is modified to count the **aircraft shortfall** at each airport in the revised flight schedule. Shortfall here refers to the difference between the original and revised number of aircraft that end at each airport.

In table 7.1, the total shortfall is listed for each of the problem instances described in table 6.1. The objective function, costs, parameters and settings are the same as those used in the RSALS-heuristic. The only difference is that 5 iterations are made.

As can be seen in table 7.1 there is an aircraft shortfall in most problem instances. There also seems to be a correlation between the size of the problem and the size of the shortfall, which is reasonable to expect. A flight planner would probably consider most of these revised flight schedules to be of poor quality. However, the size of the aircraft shortfall is – as explained – directly associated with the number of cancellations. Therefore, if the cancellation cost is increased, fewer cancellations should appear along with a decreased aircraft shortfall. The cancellation cost is therefore increased from 1.5 to 2.5 times the flight revenue and all problem instances are solved again using the exact same approach as before. The result can be seen in table 7.2.

As shown, the total aircraft shortfall decreases from 80 to 61 when the cancellation cost is increased. Whether these revised flight schedules are more acceptable than those in table 7.1 is very difficult to estimate. Flight planners might actually be willing to accept the shortfall in table 7.2 if the associated flight schedule resolves a large number of problems. They may also just prefer that cancellations are made so that aircraft shortfall does not occur at all – even if this means that there are more cancellations

Instance No.	Aircraft shortfall
1	0
2	0
3	1
4	1
5	1
6	1
7	1
8	2
9	2
10	4
11	1
12	4
13	4
14	1
15	5
16	3
17	4
18	4
19	7
20	8
21	3
22	4
23	4
24	8
25	7
Total:	80

Table 7.1: *Aircraft shortfall found in 25 problem instances using the RSALS-heuristic.*

and increased delays. This will ultimately be up to the airline using a DAR-heuristic of this kind.

It should also be noted that simply increasing the cancellation cost to a point where shortfalls never occur will not work. This will in effect render the DAR-heuristic incapable of making any cancellations at all, which of course is unacceptable.

Instance No.	Aircraft shortfall
1	2
2	0
3	2
4	1
5	1
6	1
7	2
8	2
9	1
10	3
11	0
12	1
13	2
14	2
15	5
16	0
17	2
18	2
19	3
20	7
21	2
22	2
23	3
24	5
25	4
Total:	61

Table 7.2: *Aircraft shortfall found in 25 problem instances when the cancellation cost is increased.*

7.2 Flight Schedule Structure

As mentioned in section 6.4, flights are distributed equally across the airports in all the problem instances. This is not realistic from a practical point of view.

Most airlines structure their flight plans in a **hub-and-spoke system**. Here flights are concentrated around a few large airports from which a whole range of destinations are serviced. The central airport is referred to as the hub and the other destinations are the spokes. There are typically very few flights between those destinations; travelling between them will be done via the hub airport. A simple hub-and-spoke system is illustrated in figure 7.5.

Figure 7.5: *A hub-and-spoke flight schedule.*

This figure illustrates that if a person wishes to travel from A to B, this person will first have to fly to the hub airport and then to destination B. It should clear that a lot of flights are concentrated at the hub for this reason.

The concentration of flights around a few hub airports increases the aircraft density at these airports (see section 6.4.2) and consequently the neighborhood size. The local search will therefore have to search through a greater number of solutions, requiring increased computational time. It is therefore expected that a hub-and-spoke flight schedule structure will increase the time it takes to find a local optimum using the SALS or RSALS.

A number of problem instances were generated where 50% of all flights were concentrated around three airports. These problem instances were similar in size to those in table 6.1. The computational time needed to find one local optimum was 2-4 times greater than the time it took to find a local optimum in the problem instances with an even flight distribution. The average computational time needed to find local optima in these problem instances was 2.22 seconds (see table 6.5). It is therefore reasonable to expect an average increase in computational time to approximately 10 seconds given hub-and-spoke flight schedule structure.

Another important influence that the flight schedule structure has on a DAR-heuristic involves cancellations. In hub-and-spoke systems links are typically made up of flights between the same two airports, namely the hub airport and a spoke airport. This is illustrated in figure 7.6.

Figure 7.6: *A typical link in hub-and-spoke flight schedule systems.*

As mentioned in section 7.1 some cancellations can be made which do not cause aircraft imbalance to occur. In figure 7.6 flights 1, 2 and 3 can all be cancelled while maintaining aircraft balance. Suppose aircraft 1 is very delayed, hence delaying the entire link significantly. A flight controller would probably choose to simply cancel flight 1 and 6 so that the aircraft could "catch up" with the remaining flights in the link. It would of course be preferable if the DAR-heuristic behaved the same way.

If a flight flight is delayed beyond a certain time limit it will be cancelled by the DAR-heuristic. If it is the first flight in a link which is delayed, all down-line flights will also be delayed. In that case, the heuristic will always choose to cancel the first flight, thus cancelling the entire link. By doing so, it removes the largest possible delay by making only *one* swap (see figure 7.7).

Figure 7.7: *One swap which cancels the entire original link.*

Revenue of cancelled flights does not add to the objective function, so the heuristic will try to "un-cancel" some of the flights just cancelled. Suppose the delay of each flight becomes acceptable if aircraft 1 is assigned to flight 2. This corresponds to a swap between aircraft 1 and 2 and by making this swap all flights except 1 and 6 are again active. This is illustrated in figure 7.8.

In other words, if the cancellations that do not cause imbalances are allowed and the flight schedule has a hub-and-spoke structure, the DAR-heuristic can be expected to arrive at solutions which are in line with common flight control principles. Indeed, it seems that a hub-and-spoke structure makes cancellations much easier to handle while taking aircraft balance into account.

Figure 7.8: *One swap restores 4 flights and maintains aircraft balance.*

7.3 Swap Costs

In dedicated aircraft recovery literature, the concept of swap costs is often mentioned. However, the purpose of these is not easily defined and using them in a meaningful way is not as straightforward as it may seem.

Swap costs are first of all used to control the number of swaps made in a revised flight schedule. In general, changes to a flight schedule are undesirable because of the difficulties involved. A cost can be assigned to all swaps that encourages a heuristic to make as few changes as possible. Unfortunately, using swap costs is not that simple. Suppose an aircraft is swapped to a new link and later swapped back again to its original link. This is exactly what happens in figure 7.7 and 7.8. The latter swap is very attractive, because it returns the aircraft to its original link (see section 7.4). Therefore, it does not make sense to punish the latter swap with a swap cost. On the contrary, a swap cost should – if anything – encourage the latter swap.

Consequently, the challenge with regard to swap costs is to discourage swaps in general while encouraging swaps that return an aircraft to its original link. Applying costs uniformly to all swaps will not achieve this.

A possible method of dealing with this challenge is to update swap costs every time the flight schedule is changed. This may be possible to do automatically, but it will make the heuristic slower. It would instead be faster if swap costs were static. This is possible if swap costs are a function of the actual aircraft and the flight to which it is assigned. This way, swaps that assign aircraft to *any of the flights in its original link* can be associated with zero cost. A swap cost matrix of this kind would be static, because it is based on the original links – which of course do not change. In figure 7.8, this means that the assignment of aircraft 1 to flight 2 has no swap cost. The objective function introduced in section 6.3.3 is easily modified to include the swap costs:

$$\begin{aligned} \textit{Objective} \quad = \quad & \sum_{a\in A}\sum_{f\in F} r_f \cdot x_{af} \\ & -\sum_{a\in A}\sum_{f\in F\setminus F_c} \alpha_f \cdot DF \cdot r_f \cdot d_{af} \cdot x_{af} \\ & -\sum_{a\in A}\sum_{f\in F_c} \beta_f \cdot r_f \cdot x_{af} \\ & -\sum_{a\in A}\sum_{f\in F} s_{af} \cdot x_{af} \end{aligned} \tag{7.1}$$

The parameter s_{af} was defined in section 4.6. In summary, swap costs are relatively easy to implement that discourage swaps in general while encouraging aircraft to remain/return to their original link.

7.4 Maintenance Considerations

Some aircraft have to end at a specific airport for maintenance purposes. Including such considerations will complicate the swapping procedure. In figure 7.2, a swap is made in case A that respects aircraft balance restrictions. However, the two aircraft end at new airports after the swap. Any maintenance restriction on either aircraft would therefore be violated.

Earlier in this chapter, methods to handle aircraft balance and swap costs were discussed. Both of these methods increase the likelihood that aircraft end at their original final airports. However, it may still be necessary to specifically include maintenance considerations in the DAR-heuristic. Below is a description of how this can be done.

Not all aircraft will have maintenance restrictions. The aircraft which do not have these restrictions are in principle free to end in any airport. This freedom represents combinatorial possibilities that the DAR-heuristic can use – and should use. Consequently, swaps between aircraft *without* maintenance restrictions are made in some cases even if they end in different airports.

Those aircraft *with* maintenance restrictions are now limited to swaps such as those illustrated in figure 7.7 and 7.8, i.e. swaps that do not cause an aircraft to end in new airports. However, there is one more possibility.

When swaps are made involving aircraft with maintenance restrictions, a simple search could be conducted. This search would try to find swaps further down-line in the link, which could return the aircraft to a link – possibly its own – that ended at the original final airport. This is illustrated in figure 7.9, 7.10 and 7.11

Figure 7.9: *Two aircraft with maintenance restrictions.*

Implementing the possibility of finding swaps that return aircraft to their original final airports is relatively easy. Every time a swap is made involving aircraft with maintenance restrictions, this feature is activated, so the current neighborhood definition does not need to be changed. If no swaps are found which respect the restrictions, it is simply not carried out.

Figure 7.10: *A single swap violates the maintenance restrictions because neither aircraft end at their original final airport.*

Figure 7.11: *By searching through flights further down-line in each link, a swap is identified that returns each aircraft to their original final airports.*

7.5 Airport Curfews

Most airports in Europe have **curfews**, i.e. no aircraft can land or take off for a certain period of time every night. Some flights may therefore not be allowed to take off if they are delayed: The curfew at the origin airport may be violated or the aircraft will arrive too late at its destination airport. Flights that violate airport curfews will under normal circumstances be cancelled.

Including airport curfew considerations in a DAR-heuristic is relatively easy. In its current form the DAR-heuristic re-evaluates the entire link of both involved aircraft every time a swap is made. This includes updating departure and arrival times. Given this information, it would be simple to cancel those flights in each link that violate curfew restrictions. In this way, the cost of a given swap would include the cancellation costs needed to respect curfews.

7.6 Passenger Flow

There are two aspects involved when discussing passenger flow. Firstly, aircraft may not be assigned to flights if there are more passengers than seats. Secondly, when cancelling or delaying aircraft, some passengers will miss their connecting flights further down-line.

The first aspect is easily considered in a DAR-heuristic. When an aircraft is assigned to a flight, the seat capacity is simply compared with the number of passengers. If there are too many passengers, the swap is not allowed and vice versa.

However, the second aspect is more difficult to deal with. It is most likely impossible to cancel and delay aircraft without upsetting the itinerary of some passengers. The objective would therefore be to minimize these itinerary disturbances when constructing a revised flight schedule. However, including such an objective explicitly would probably not improve the revised flight schedule significantly: Itinerary disturbances arise because of delays and cancellations, which is exactly what the DAR-heuristic attempts to minimize. In other words, the objective of minimizing itinerary disturbances is almost identical with the objectives in the current DAR-heuristic. It therefore seems reasonable not to include specific itinerary considerations at this stage in the DAR-heuristic.

7.7 Multiple fleets

Certain aircraft cannot be swapped and there may be numerous reasons for this. Some reasons that have not been mentioned yet are associated with the *type* of aircraft. Airlines typically have several **fleets** consisting of different aircraft types. Aircraft in different fleets can often not be swapped, e.g. the seating arrangements in two aircraft may be very different. Likewise, a number of other technical circumstances may prevent aircraft from being swapped. Some of these circumstances are exotic and impossible to incorporate in a heuristic because they rarely arise. Other technical circumstances are general enough to be included, e.g. the gates at which two aircraft are located may exclude the possibility of swapping them.

General technical circumstances can be considered by the heuristic. This would simply be done through swap costs like those described in section 7.3, i.e. very large swap costs can be assigned to swaps which are unacceptable. This could also be made interactive through cost multipliers, such as it was described in section 6.3.3.

7.8 Ferrying of Aircraft

Another concept mentioned often in dedicated aircraft recovery literature is **ferrying aircraft**. This feature may be a interesting from a theoretical point of view, but apparently not in normal flight control. At British Airways, ferrying is almost never considered an option due to the cost of carrying out flights with no passengers. The circumstances in which British Airways actually do ferry aircraft are always extraordinary. Therefore, including the option of ferrying in a DAR-heuristic would be virtually impossible, not to mention superfluous.

It is possible that other airlines use ferrying often enough to justify the ferrying feature. If so, the feature is relatively easy to implement: The set of candidate flights considered for each aircraft is simply augmented to include all flights, including those at other airports. However, as explained in section 6.4.2 the computational time increases when the size of the candidate sets is increased. In the extreme, all aircraft could be assigned to all flights. This would increase the size of the solution space dramatically and most likely increase the computational time accordingly (see section 6.3.1).

7.9 Conclusion of Chapter 7

To make a DAR-heuristic function in a real context, a number of considerations must be added to the heuristic implemented in chapter 6. These considerations are all discussed in this chapter and a description of how they can be included is given. It appears relatively simple to add most of these to the current DAR-heuristic. In short, the simplicity of the underlying network and the heuristic design seems to allow the necessary flexibility to add the extra considerations.

Chapter 8

The Integrated Crew and Aircraft Recovery Problem

Chapters 6 and 7 described a heuristic approach to solving DARP where only aircraft are considered. This chapter describes a possible method of integrating crew and aircraft considerations in an Integrated Crew and Aircraft Recovery Problem (ICARP) using heuristics. A more exact definition of ICARP was given in chapter 2.

8.1 Motivation

Crew and aircraft considerations with respect to disruption management are completely interconnected. Decisions concerning only one of the two will most likely affect the other, i.e. rotating crews might delay an aircraft if the relevant crews are not made available on time. Conversely, rotating aircraft might delay crews, which again may delay other aircraft. In other words, the DAR-heuristic described in the previous two chapters can only provide a revised flight schedule, after which flight planners have to construct new crew pairings and rosters. For this reason, finding methods for considering crew *and* aircraft simultaneously will ultimately help provide flight planners with a revised aircraft and crew schedule (henceforth referred to as the flight schedule), thus increasing the speed with which they can repair the schedule disruptions.

8.2 Goals for Chapter 8

The purpose of this chapter is to describe a method of including both crew and aircraft considerations in an ICAR-heuristic.

8.3 Definition of Network

The ICAR-heuristic is based on an underlying network (see figure 8.1). This network is identical with the DARP-network in figure 6.1 except for the addition of crew nodes. In other words, all features pertaining to aircraft and flights remain unchanged in relation to their definition given in chapter 6.

Notice that all the flight nodes are placed at the point in time when they *should* have left, whereas the aircraft nodes are placed at the point in time where they are *actually* ready. Hence, an upward arc indicates a delay. Notice also that this is not true for the crew nodes. These are placed close to the aircraft nodes, from which the relevant crew disembarks, but their location does *not* indicate the time at which the crew units are ready.

The crew nodes in figure 8.1 are simplified representations. In fact each crew node holds information about the assignment of each **crew unit**. Units are one or more crew members of the same **crew type**. There are several types of crew, i.e. captains, 1st pilots, stewardesses, etc.. A certain number of units of each type are required to operate an aircraft. However, the whole crew on an aircraft does not necessarily stay together during the entire working day. In principle, each individual crew unit could be assigned to a distinctly different pairing. For example, 2 stewardesses who are both assigned to flight 1 in figure 8.1 are not necessarily both assigned flight 5 at airport 2.

Crew nodes are placed so that they are close to the aircraft nodes, from which the relevant crew units disembark, but the location does *not* indicate the time at which the crew units are ready.

In this chapter, it is assumed that only 2 crew types exist, namely cockpit crew and cabin crew. It is furthermore assumed that 1 cockpit crew unit and 2 cabin crew units are needed to operate an aircraft. In figure 8.2, a closeup of figure 8.1 is shown to exemplify the underlying crew node design.

Figure 8.1: *Underlying network to use with an ICAR-heuristic that includes both aircraft and crew considerations*

As can be seen in figure 8.2, the crew node actually consists of the 3 crew units required to operate the aircraft. The arcs pointing to these unit nodes will always come from the same aircraft – the logic being that once the aircraft has landed and the crew units have disembarked, they are all placed in the crew node symbolizing their availability. In the case of crew *supply* nodes, there is no upper bound on the number of crew units. All crew units that initiate their pairing from a particular airport are placed in the supply node. Likewise, depending on the number of flights departing from a particular airport, a suitable number of standby crew units of all types are placed here. Notice that crew units are always assigned to flights – never to aircraft.

Figure 8.2: *Closeup of figure 8.1 to illustrate the exact crew node design.*

In figure 8.1, crew pairings are ultimately assigned to sink nodes just like aircraft. Each individual crew unit is supposed to end at a certain airport at the end of a working day. By using the sink nodes, each crew unit can be tracked to its final destination. It is a specific goal of the heuristic that as many of the crew units as possible end at their original final destinations. By ending each crew unit in a sink node, this can be achieved.

An important difference between aircraft links and crew pairings should be noted here. As discussed in chapter 2, when aircraft are rotated it is less important that an aircraft ends at its original final destination as long as some other aircraft does. More importantly, a certain *number* of aircraft should end at a particular airport. With crew units a similar but tighter restriction exits. Here the specific crew unit has to end at a specific airport. As will be discussed in this chapter, this limits the possibilities of rotating crew significantly and thus, makes the dedicated crew recovery problem more complex and difficult to solve.

8.4 Outline of the ICAR-heuristic

The ICAR-heuristic can be summed up as follows: (i) Optimize the flight schedule with respect to aircraft only; (ii) Modify the crew pairings and rotations to accommodate the revised flight schedule found in the previous step; (iii) Repeat this process to produce different revised flight schedules.

The idea is that the SALS heuristic used to solve DARP in chapter 6 is applied to step (i) above. SALS is extremely fast when applied to DARP so this hopefully leaves enough time to find suitable crew pairings and rosters in step (ii). A general outline of the ICAR-heuristic is shown in figure 8.3. Note that x refers to a complete flight schedule.

```
procedure ICARP
    $x_0$ = GenerateInitialSolution
    repeat
        $x$ = ModifyAircraftAssignments($x_0$)
        repeat
            $x'$ = LocalSearchAircraft($x$)
            $x$ = AcceptanceCriterion1($x, x'$)
        until A better solution cannot be reached by
            rotating aircraft only.
        repeat
            $CrewType$ = SelectCrewType()
            IdentifyPointsOfConflict($x, CrewType$)
            repeat
                $x'$ = ResolveConflicts($x, CrewType$)
                $x$ = AcceptanceCriterion2($x, x'$)
            until A better solution cannot be reached by
                rotating this type of crew only.
        until All crew type pairings have been optimized.
        $x^*$ = AcceptanceCriterion3($x, x^*$)
    until Stopping criterion is met
end ICARP
```

Figure 8.3: *Outline of the ICAR-heuristic procedure.*

As can be seen from the ICAR-heuristic illustrated in figure 8.3, there are 9 main elements. These are:

GenerateInitialSolution: The initial solution consists of the original flight schedule including any delays/cancellations that may exist.

ModifyAircraftAssignments: This procedure modifies aircraft assignments in the original schedule by making a number of random swaps. This assures a different starting point for each iteration. An iteration is considered to be exactly one pass through the outermost loop in figure 8.3.

LocalSearchAircraft: This procedure finds the best local optimum in the neighborhood of the current solution/schedule with respect to aircraft only.

AcceptanceCriterion1: Same as AcceptanceCriterion described for SALS earlier (see section 6.8.2).

SelectCrewType: This function selects a crew type, whose pairings and rosters have not yet been optimized, and returns this type to the variable $CrewType$.

IdentifyPointsOfConflict: Once a local optimum has been found considering aircraft only, there will be conflicting crew assignments which have to be resolved. This function identifies these conflicts for each crew type.

ResolveConflicts: While the aircraft roster is retained, this function attempts to resolve all points of conflict for each crew type.

AcceptanceCriterion2: If a feasible or better crew pairing modification is found, this function returns the revised solution.

AcceptanceCriterion3: This function ensures that the best solution found overall is stored. This is again done through a function like that shown in equation 6.4.

Aside from *IdentifyPointsOfConflict*, *ResolveConflicts*, *AcceptanceCriterion2* and *SelectCrewType* the functions described above are basically the same as those described for the SALS heuristic in section 6.8. The following section describes the general idea behind these 4 functions and the how they are integrated in the ICAR-heuristic.

8.5 The ICAR-heuristic – In Words

This section is devoted to explaining the ideas behind the outline illustrated in figure 8.3. This outline can also be formulated in words, as it is done in figure 8.4. The remaining part of section 8.5 will describe each of the steps here in detail.

1. Generate an initial flight schedule.
2. Optimize the flight schedule with respect to aircraft only.
3. For each crew type, find all points of scheduling conflicts.
4. Resolve the conflicts to the furthest extent possible.
5. Repeat these steps for different initial solutions.

Figure 8.4: *ICAR-heuristic outline - in words.*

8.5.1 Step 1: Initial Solution

The initial solution used by the ICAR-heuristic is the original flight schedule including all delays and cancellations of crew and aircraft. This means that all the aircraft to flight assignments and crew to flight assignments are retained in the original schedule – even if the original schedule is infeasible. This approach allows the heuristic to find revised flight schedules, which differ as little as possible from the original schedule.

8.5.2 Step 2: Dedicated Aircraft Recovery

The ICAR-heuristic outline is based on the assumption that resolving aircraft disruptions is a first priority, followed by resolving crew disruptions.

In theory there are 2 other approaches. The most obvious one is that the heuristic could attempt to consider crew and aircraft rotations simultaneously. However, the complexity of doing so is vast due to the endless number of combinatorial possibilities – most of which will result in infeasible flight schedules. It is also difficult to define a meaningful neighborhood given the current heuristic design, because aircraft, cockpit crew and cabin crew cannot be swapped indiscriminately. Possibly, the use of constraint programming could be used to construct neighborhoods consisting of feasible solutions, but certainly this would be a difficult task. Finally, evaluating solutions is time consuming, especially if both the aircraft *and* crew assignments are evaluated. Given the number of combinatorial possibilities and the evaluation time, considering both aircraft and crew assignments simultaneously, seems a very difficult approach to solving ICARP – certainly within the framework used in this thesis.

The other alternative approach is, of course, to consider crew assignments *before* aircraft assignments such as Teodorović and Stojković (1995) did in their attempt to solve ICARP. Here they argued that this approach resulted in faster solution time. However, it seems more reasonable to resolve the problem with the least options first, which is why their approach seems unattractive. Flight planners have to resolve aircraft disruptions by rotating aircraft that are already in use. Airlines rarely have a surplus aircraft due to the costs of having one, so using such an aircraft to resolve a disruption is almost never an option. For similar reasons, dead-heading aircraft is not an option that airlines use. The only **3** options are therefore to delay, cancel and swap aircraft. When crew disruptions are resolved, there are **5** options available: delaying, placing crew on standby, swapping, using reserve crews and dead-heading crews. In other words, flight planners have more options when they try to resolve crew disruptions than aircraft disruptions. It therefore makes sense to resolve the aircraft disruption first and then see if feasible crew rosters/pairings can be made on that basis.

Teodorović and Stojković (1995) may be correct that rotating crew before aircraft results in faster solution times. However, their method of doing so leaves a lot of unanswered questions. In particular it is not explained how they retain as much of the original schedule as possible – one of the most important quality parameters in crew and aircraft recovery for European airlines. Likewise, the extent to which their algorithm is able to minimize the number of cancellations and the delays is not discussed explicitly – it is only mentioned that these are the main priority. For these reasons, it is very difficult to evaluate the work of Teodorović and Stojković (1995). Despite their conclusions, it is reasonable to assume that considering aircraft before crew can lead to a fast ICAR-heuristic when considering the successful implementation of the DAR-heuristic in chapter 6.

At British Airways they typically consider aircraft before crew when disruptions occur. The disruptions are resolved for aircraft first after which a suitable crew solution is found. If a revised crew roster cannot be found, a different solution is found for the aircraft disruption and the pattern is repeated. Naturally some disruptions occur which involve crew only, i.e. if one crew is delayed, a reserve crew can be used instead and the flight plan remains unchanged. However, in such cases a mathematical tool like a heuristic may be superfluous and the disruption can instead be resolved manually by a flight planner. This heuristic is designed to resolve those disruptions, which cause a series of delays/cancellations due to the interconnection between aircraft, flights and crews.

Initially a SALS heuristic exactly like the one in section 6.8 is used to find a local optimum for the aircraft assignments only. Throughout SALS the crew units remain assigned to their original flights – even if the flights are significantly delayed or even cancelled.

8.5.3 Step 3: Identify Points of Conflict

Four types of conflicts may arise during step 2 in the ICAR-heuristic procedure. These conflicts are:

1. Crew may be assigned to flights, which have been cancelled.
2. Some flights may have an insufficient number of crew units assigned.
3. After the aircraft optimization, some crew assignments may cause flights to be delayed.
4. Some crew pairings may conflict with restrictions imposed by unions or general regulations.

These 4 types of conflicts are each described below.

Type 1 Conflict: Cancelled Flights

Figure 8.5 illustrates a flight schedule segment before and after the dedicated aircraft recovery of step 2. It can be seen that aircraft 5^* has been delayed in the original schedule. After the aircraft optimization, aircraft 5^* is assigned to the sink node and no other aircraft is assigned to flight 5. This means that flight 5 is cancelled, yet the incoming crew on aircraft 5^* is still assigned to flight 5.

Type 2 Conflict: Crew Shortage

Figure 8.6 illustrates the same segment as in figure 8.5. However, after the dedicated aircraft recovery, the incoming flight to aircraft node 5^* has

Figure 8.5: *Illustration of conflict type 1.*

been cancelled and some other aircraft has been assigned to flight 5. Just as before, the crew from aircraft 5* is still assigned to flight 5. This is impossible, because the incoming flight was cancelled leaving no crew to assign. Nevertheless, the assignment remains – this is called an *empty crew assignment*. The node to which this empty crew is assigned is called an *empty crew node*. Delayed crew units are not considered as crew shortage at this point in the heuristic, irrespective of the delay size.

Type 3 Conflict: Delays Caused by Crew Assignments

Assume that figure 8.7 again illustrates a small segment of a schedule before and after the dedicated aircraft recovery. Before DAR aircraft 5* is assigned to flight 5 and likewise for aircraft 6 and flight 6. However, aircraft 5* is delayed, which again causes flight 5 to be delayed. After DAR, aircraft 5* and 6 have been swapped, but the crews have not. The result is that despite aircraft 5* and 6 having been swapped, flight 5 is still delayed as much as before, but now flight 6 is also delayed.

Figure 8.6: *Illustration of conflict type 2.*

Type 4 Conflict: Crew Restriction Violations

The final type of conflict involve pairings that violate restrictions on crew units. This type of conflict is most likely present even before DARP is solved due to the delays/cancellations in the original schedule.

There are numerous crew restrictions. For example, if a standby crew unit was placed in the crew supply node because its flight was cancelled (see figure 8.8), this crew unit will have to end at the airport where it was originally supposed to end.

Other restrictions exist that are similar for all crew units, i.e. the length of a pairing measured in time must not exceed a certain limit. At British Airways for example, if a crew member has been on standby for 5 hours, its pairing length must not exceed 7 hours and 30 minutes (see British Airways (2000)). Likewise, there are tight restrictions on the crew unit turn-arounds, maximum number of flights in a pairing, etc..

It should be clear that a vast number of restrictions on crew units exist.

Figure 8.7: *Illustration of conflict type 3.*

In order to deal effectively with the complexity of these restrictions the ICAR-heuristic places all restrictions into 2 categories:

Hard restrictions: These restrictions must be respected entirely. A violation of these renders the flight schedule infeasible.

Soft restrictions: These are restrictions that can be violated if absolutely no other options exist.

Hard restrictions include those described above, i.e. restrictions on pairing length, **crew turn-around** time, number of flights in a pairing, crew unit certification, etc. The soft restrictions include crew assignments that cause delays, minor crew shortages, which languages the crew units speak, etc.. An exact listing of how crew unit restrictions are categorized will ultimately have to be decided on by the airline using the ICAR-heuristic.

8.5.4 Step 4: Resolving Conflicts

The key to an ICAR-heuristic lies in how the before mentioned conflicts are resolved. Conflicts 1 and 2 are resolved in that order, however, conflicts

3 and 4 are resolved simultaneously. Below follows a description of how each conflict is attempted resolved and the guiding principles behind every decision possibility.

Resolving Type 1 Conflicts

Type 1 conflicts – crew assigned to cancelled flights – occur every time a flight has been cancelled. The way the ICAR-heuristic resolves this conflict is by assigning the crew to the crew supply node at the airport from which the cancelled flight should have left according to the original schedule. This is illustrated in figure 8.8.

Figure 8.8: *Resolving conflict type 1.*

By reassigning crew that were assigned to cancelled flights to the crew surplus node, 2 important things are achieved: Firstly, this crew can be made available to other flights. Secondly, this crew can be dead-headed to their final destination if they remain unassigned. It should be noticed that all type 1 conflicts can be resolved without exception, because resolving it is a simple matter of reassigning the crew.

Resolving Type 2 Conflicts

Type 2 conflicts – crew shortages – are reasonably straightforward to resolve. These conflicts arise when dedicated aircraft recovery has taken place and all type 1 conflicts have been resolved. Unlike type 1 conflicts, there is no guarantee that all crew shortages can be resolved. Instead these conflicts are resolved to the furthest extent possible.

Type 2 conflicts arise on 2 occasions: Firstly, if a flight has been cancelled then all down-line flights, where crew from the cancelled flight were supposed to have been assigned, will have a crew shortage. Secondly, if a crew member fails to show up, e.g. because of illness, the flight to which this crew member was assigned will have a crew shortage – including all the down-line flights in the crew pairing. The general method of resolving type 2 conflicts for each crew type is listed below:

1. Identify all flights, where a crew shortage exists and retain all the associated empty crew assignments.
2. Find the earliest flight, with a crew shortage.
3. Examine if standby crew exists in the crew supply node at the same airport, who can be assigned to this flight.
4. If such crew units exist, assign them to the flight in question.
5. Update the flight schedule.
6. Repeat until no crew shortages exist or no further crew shortages can be repaired.

These steps are simple statements of what actually happens when type 2 conflicts are resolved. Below follows a more detailed description of each step.

Step 1

The primary function of this step is simply to identify the flights with crew shortages. However, it is important that the *empty crew assignments* are retained in this process.

There is a very important reason behind retaining these assignments: They represent original *legal* pairings (or parts thereof) where all rules, regulations, preferences, etc. have been considered and included. Therefore, if at all possible, these pairings should be retained. Retaining the empty assignments also enables the heuristic to evaluate the down-line effects of assigning standby crews to flights with crew shortages (see step 2).

Step 2

Once the flights with crew shortages have been identified, the order in which they are resolved becomes important. This is illustrated in figure 8.9.

Figure 8.9: *The order in which type 2 conflicts are resolved is important. In case A, all conflicts are resolve through one crew assignment where only one conflict is resolved in case B. The remaining conflicts in case B will have to be resolved through other crew assignments.*

In this simplified schedule, the crew units originally assigned to flight 4 had a pairing identical with the link undertaken by aircraft 4. Suppose a disruption has occurred, which results in flight 4 having no crew units assigned to it. As a consequence, there is also no crew available for each of the down-line flights. Suppose also that sufficient standby crew exist at both airport 1 and 2. It is then clear that the standby crew assignment in case 1 is better than in case 2.

When a standby crew is assigned to flight 4, as in the first case in figure 8.9, this crew can undertake the entire pairing, thus eliminating the need for more standby crew. However, if a standby crew from airport 1 is assigned flight 2, as in the second case, the crew shortage will *only* be resolved for that flight. At least one more complement of standby crew units will therefore be needed for the flights further up-line in the link. For this reason, the earliest crew shortages should generally be resolved first. Other strategies could be attempted, for instance: Select an airport at random and resolve the earliest crew shortage here. Whether or not this strategy will work better is difficult to estimate without empirical evidence, but the earliest-crew-shortage-first-strategy seems most logical.

Step 3

The purpose of step 3 is to identify crew units that can man the earliest flight with a crew shortage taking into consideration the *hard* restrictions only. Such crew units are referred to as *eligible* and are searched for in the crew supply node (see figure 8.1). Note that such crew units do not necessarily exist for each unmanned flight. Conversely, there may be more eligible crew units than necessary. The soft restrictions are not considered explicitly until type 3 and 4 conflicts are resolved.

Step 4

The result of step 4 depends on the three possible outcomes of step 3:

- No eligible crew units are found.
  - In this case, the flight is simply left unmanned. Later adjustments may resolve this crew shortage through dead-heading crew or cancelling the flight.
- The exact number of eligible crew units needed are found.
  - The crew units are assigned to the flight.
- Several eligible crew units are found, which could resolve the crew shortage.
  - In this case, the heuristic selects the crew unit, whose assignment to the flight in question will resolve the greatest number of down-line crew shortages, cause the least delays and has been on standby for the longest period of time.
  - There is a trade-off between these 3 priorities that an airline using this heuristic would have to quantify.

Step 5

Step 5 consists of updating the flight schedule with the crew unit assignments made in step 4. This is illustrated in figure 8.10. Assume that the eligible crew units are assigned to flight 4. After the update there no longer exists any type 2 conflicts, so when the heuristic continues to resolve crew shortages, it will no longer have to consider these – and that is the purpose of step 5. The example used here is, of course, based on the assumption that none of the hard restrictions are violated at any time in the pairing.

Figure 8.10: *Updating flight schedule after crew assignment.*

Step 6

The purpose of step 6 is to ensure that all crew shortages are resolved to the extent possible before the heuristic continues. It therefore repeats the 5 steps just described until this is achieved.

Resolving Type 3 and 4 Conflicts

This is the final part of the ICAR-heuristic. Here the delays caused by crew assignments and crew restriction violations (type 3 and 4 conflicts respectively) are dealt with simultaneously. The reason for this is that these 2 types of conflict are completely interdependent. Situations will arise, where type 3 and 4 conflicts are mutually inclusive, i.e. to avoid one the other must appear. Where hard restrictions are *not* involved, there will be a trade-off between the two conflicts and the priorities involved here must ultimately be decided on by the airline using this heuristic.

It is important to understand this stage of the flight schedule. The original flight schedule has a number of delays/cancellations. After the dedicated aircraft recovery, some of these delays have most likely been reduced and the number of cancellations may also have changed. As a result, some crew pairings have become legal – others illegal. In other words there are a series of problems that have to be resolved and because they are interdependent, a priority list of which problems to resolve first is listed below.

Problem 1: Violations of hard restrictions.

Problem 2: Unacceptable crew shortages on flights which were not cancelled during the dedicated aircraft recovery.

Problem 3: Crew assignments that cause delays.

Problem 4: Violations of soft restrictions.

How these problems are resolved will be described in detail later. However, there is a general principle that applies to resolving any of the 4 problems: For each problem the ICAR-heuristic will initially identify all flights where the particular problem occurs. It will then attempt to resolve these problems starting with the *earliest* problem first. This principle was described in section 8.5.4 and illustrated in figure 8.9.

For each item in the problem list above, there are a series of possible solutions methods. However, which of these to apply is also a matter of priority. These solution methods are listed below in their order of priority:

1. Swapping active crew units.
2. Swapping active crew and standby crew units at the same airport.
3. Dead-heading standby crew.
4. Delaying flights.

5. Cancelling flights.

The application of these solutions to each of the problems requires a comprehensive explanation.

Solving Problem 1

It is imperative that the violations of hard restrictions are resolved completely. This is done by applying each of the solution methods to the problems in their order of priority.

Applying Solution Method 1 – Swapping Active Crew Units

The first solution method involves swapping active crew units. This cannot be done indiscriminately, so the concept of *re-linking* is introduced (see Larsen et al. (2001)). Re-linking is illustrated in figure 8.11.

In step 1 of figure 8.11, pairing A violates a hard restriction – namely the minimum turn-around time is not respected. Assume that both flight $a2$ and $b2$ originate at airport 1 and end at airport 2. In step 2, therefore, crew A and B are swapped so that the minimum turn-around time is respected for both crews. However, neither crew ends at the airport where they were originally supposed to end – hard restriction violation 2. The last step therefore involves swapping the crews back to their original pairing, thus resolving all hard restriction violations. This particular re-link could be described as a 2-swap, because 2 pairings were involved. However, situations where swaps (re-links) between 3 or more pairings resolved restriction violations are certainly possible - although difficult to implement.

It is difficult to estimate how many conflicts can be resolved using re-linking. However, it has been demonstrated that re-linking can be used successfully on a dedicated crew recovery problem where real data from British Airways was used (see Larsen et al. (2001)).

In their work, the flight schedule is static with respect to aircraft. In other words, no crew reassignments are made that cause delays. Conversely, should crew reassignments be made so that flights become less delayed – **rolled up** – this is not utilized. The ICAR-heuristic *could* make use of this. To understand this, refer to figure 8.7. Here both flights are delayed as a consequence of crew assignments. Assume that the two crew units can be swapped such as it is shown in figure 8.12.

Figure 8.11: *Illustration of the re-linking concept.*

Figure 8.12 illustrates that sometimes it is possible to make swaps that reduce delays. Naturally, these should be utilized, so anytime where the ICAR-heuristic makes a feasible swap that achieves this, it will retain this assignment – even if it does not specifically solve a problem (in this case a hard restriction violation).

Applying Solution Method 2 – Using Standby Crew

If hard restriction violations still exist after the application of re-linking, the ICAR-heuristic will attempt to use standby-crew from the same airport where the violation has occurred. In basic terms, this involves swapping

Figure 8.12: *Making delays smaller by making a crew swap.*

standby crew units with the crew units, whose pairing has a hard restriction violation. The replaced crew is by default assigned to the supply node. Here they may be assigned to new flights if no restrictions are violated, or they may simply be dead-headed to their final destination.

Applying Solution Method 3 – Dead-heading Crew

This solution method is very similar to the previous one. The only difference is that in this case standby crew units do not come from the airport where the hard restriction violation has occurred. Instead, eligible crew units are dead-headed to the airport where they are needed. The crew units that they replace are again assigned to the supply node at the airport where the violation occurred.

Dead-heading crew is also used to ensure that crew units end at their original final destination. In other words, if a complement of crew units were delayed and corresponding standby crew units replaced them, then the delayed units would no longer be assigned to the pairing which returned them to their base station. In such case, dead-heading is used to return them if possible.

Applying Solution Method 4 – Delaying Flights

If the above 3 methods do not resolve all hard restriction violations, then further flight delays are made acceptable. This means that the above 3 methods are reapplied, but this time crew unit assignments that cause delays are permitted.

Delaying flights may in itself resolve some hard restriction violations. For example, if a crew is assigned to a flight that leaves before their turn-around time is over, the flight can simply be delayed, thus resolving the crew violation. This is illustrated in figure 8.13. Crew violations that can be resolved this way are of course resolved before repeating any of solution methods 1 through 3.

Figure 8.13: *Resolving hard restriction violations by delaying flights.*

Figure 8.13 also illustrates another important challenge. When delays are used to resolve hard restriction violations, there may be serious down-line effects. For instance, delaying one flight may result in delaying all down-line flights. This situation may be aggravated if pairing lengths as a consequence become too long, thus creating new hard restriction violations.

Real flight schedules have a certain amount of slack built into them. For this reason, it may be possible to delay a flight – including all affected down-line flights – without violating any new hard restrictions. If delaying all affected down-line flights does cause new hard restriction violations, these can possibly be resolved by repeating solution methods 1 through 3. If that does not work, then one or more flights have to be cancelled.

Applying Solution Method 5 – Cancelling Flights

If none of the 4 previous solution methods are able to resolve a hard restriction violation, the only option left is to cancel one or more flights. In this case, there is no obvious order in which to cancel flights that are a part of those pairings with hard restriction violations. From the point of view of passengers, it is most convenient if cancellations take place as far into the future as possible. This way passengers will have time to change their plans or the airline can possibly arrange other travel arrangements for them. However, passenger convenience may not be a priority if the point is reached, where cancelling flights is the only option left. The priorities in such a situation will be:

1. Minimize the number of cancellations needed to resolve the hard restriction violation. Cancellations will most likely involve a set of flights, i.e. **here-and-back** trips in a hub-and-spoke system, because such cancellations implicitly take crew considerations into account.
2. If several possible solutions exist involving the same number of cancellations, select the solution that causes the fewest *new* hard restriction violations.
3. Select the solution with the latest cancellation(s).

It is important to realize that a cancellation almost always will cause new hard restriction violations, in particular, crew units will end at the wrong airports. The idea is that these can be resolved by repeating the 4 previous solution methods. It is also important to realize, that some feasible solution can *always* be found. In theory, all flights can be cancelled thus leaving a feasible – but very unattractive – flight schedule.

Solving Problems 2, 3 and 4

The 5 solution methods applied to solve problem 1 work in exactly the same way when applied to the other three problems. These are consequently resolved to the furthest extent possible in their listing order. In using any of the solution methods to do this, the heuristic *must not* make flight schedule changes that cause an earlier problem to arise again. For example, when the heuristic makes flight schedule changes to resolve problem 2, those changes must not cause problem 1 to arise again.

8.5.5 Step 5: Repeat the Process

This is the final step of the ICAR-heuristic. At this point there is a revised flight schedule where no infeasibilities exist. However, there may, of course, be flight schedules that are even better. This step retains the best solution and repeats the algorithm for different initial solutions until some criterion is met, e.g. a time limit. Flight controllers would like several different revised flight schedules to choose from and this function is easily incorporated into this step. A series of criteria could be set up and the best solution with respect to each of these could be saved. This way, solutions with for example the smallest delay, fewest cancellations, best aircraft balance, etc. can be retained.

8.6 Complexity of the ICAR-heuristic

The complexity of the ICAR-heuristic is difficult to estimate properly because the running time depends on the number of conflicts and crew types involved. The ICAR-heuristic is in essence initiated by the DAR-heuristic, which had a complexity of $\mathrm{O}(n^3)$, where n is the number of flights (see section 6.3.1). Since DARP and the remaining part of the ICAR-heuristic are solved separately, it is interesting if the complexity of the latter part exceeds that of the DAR-heuristic.

The latter part of the ICAR-heuristic consists of three main activities: (i) Identifying crew conflicts; (ii) Resolving those conflicts; (iii) Updating the flight schedule accordingly. In (i), all crew/flight nodes are traversed to locate the conflicts. Likewise in (ii) where finding suitable crew units to resolve the conflict could require an inspection of all crew/flight nodes. Finally in (iii), the schedule must be updated, which in theory requires that all crew and flight nodes are traversed again. Activity (i) is repeated once for every crew type and activity (ii) and (iii) once for every conflict. If it is assumed that the number of crew types is constant with regard to the complexity, then the complexity of the the latter part of the ICAR-heuristic becomes $\mathrm{O}(number\ of\ conflicts \cdot n)$. The total ICAR-heuristic complexity therefore becomes the greater of $\mathrm{O}(n^3)$ and $\mathrm{O}(number\ of\ conflicts\ \cdot\ n)$. However, the number of conflicts will most likely not exceed n^2 – that would correspond to 2 conflicts for every flight. Therefore, it seems reasonable that the ICAR-heuristic has the same complexity as the DAR-heuristic, namely $\mathrm{O}(n^3)$.

8.7 ICAR-heuristic Example

This section is devoted to illustrating how the ICAR-heuristic works on an actual test instance. To do this, a small test instance has been generated where some of the situations described in the previous section can be illustrated. Not everything can be illustrated, because a test instance where all types of problems and solution methods become relevant is difficult to create – and most likely very large. The example below follows the exact steps discussed previously.

8.7.1 Step 1: The Initial Schedule

This initial schedule consists of 3 aircraft that service 12 flights between 3 airports. The original links carried out by the 3 aircraft look as follows:

```
-----------------------------------------------------------------------------
Flight Scheduled Actual  Assignments
 link    take-   Ready-
          off     time
-----------------------------------------------------------------------------
   1       56      354    (12-12) -> (2-2) -> (15-15) -> (5-5) -> (17-18)
   2       36       36    (11-11) -> (7-7) -> (14-14) -> (4-4) -> (16-18)
   3       25       25    (10-10) -> (1-1) -> (13-13) -> (3-3) -> (8-9)
-----------------------------------------------------------------------------
```

Flight link 1 should have left 56 minutes from the decision point, but the aircraft is delayed, hence the first flight in link 1 is delayed to 354 minutes from the decision point. This means that each of the down-line flights also are delayed by 298 minutes. Link 1 is illustrated in figure 8.14.

In figure 8.14, all the crew units remain assigned to the flights which are a part of their original pairings. All the crew units originate in supply node 15 and are initially assigned to flight 12. They then follow the link undertaken by aircraft 12 and end back at airport 3 approximately 800 minutes from the decision point. They should have arrived back at airport 3 approximately 500 minutes from the decision point, but the delay of aircraft 12 causes them to be delayed as well.

Unlike link 1, link 3 is not delayed at all. This link is illustrated in figure 8.15. Here, some crew units do not follow the entire link. All the crew units originating in supply node 15 undertake flight 10 and 1 in that order.

Figure 8.14: *Original link 1.*

However, at airport 3 a crew unit disembarks and is replaced by a new crew unit from the supply node. This is also the case at airport 1 after flight 13.

A figure illustrating the entire schedule with all flights and crew unit assignments is very congested and will not be shown. Link 2 is not shown,

Figure 8.15: *Original link 3.*

because it does not illustrate anything new.

8.7.2 Step 2: Dedicated Aircraft Recovery

Following the heuristic outline described in figure 8.3 the flight schedule is now optimized until a better solution cannot be reached by only rotating aircraft. This results in the following flight links:

```
----------------------------------------------------------------------------
Flight Scheduled Actual  Assignments
 link    take-   Ready-
          off     time
----------------------------------------------------------------------------
   1      276      354    (12-15) -> (5-5) -> (17-18)
   2       36       36    (11-11) -> (7-7) -> (14-14) -> (4-4) -> (16-18)
   3       25       25    (10-10) -> (1-2) -> (15-13) -> (3-3) -> (8-9)
----------------------------------------------------------------------------
------------------
CANCELLED FLIGHTS:
------------------
Flight No: 12
Flight No: 1
------------------
```

Flights 12 and 1 (in original links 1 and 3 respectively) have been cancelled. Flight 1 was not delayed at all in the original schedule. However, it may be better to cancel this flight if, for example, flight 2 can be undertaken instead with no delay at all. In any event, the heuristic found this combination of swaps, delays and cancellations to be the best possible given the set of parameters discussed in section 6.3.2.

8.7.3 Steps 3 and 4: Identifying and Resolving Points of Conflict

Step 3 is the first step that is repeated several times during the ICAR-heuristic. It identifies the different types of conflicts in the order they are resolved. The reason is that none of the conflicts are static; depending on the choices made while running, new conflicts may arise. Step 4 then resolves the conflicts in the same order that step 3 finds them.

Type 1 Conflicts

Type 1 conflicts are identified first (crew assigned to cancelled flights). These points of conflict are located by simply marking all those flights to which no aircraft are assigned. In this example there are two such occurrences, which are both resolved by reassigning the crews to the supply node. This is illustrated in figure

8.16.

Figure 8.16: *Type 1 conflicts: Flights 1 and 12 are cancelled, so the crew units are reassigned to the supply node.*

Type 2 Conflicts

Type 2 conflicts (crew shortages) occur when incoming flights are cancelled or if crew units become unavailable. Type 2 conflicts are also easy to identify. Unless caused by crew units failing to show up, they will always occur down-line from a cancellation. The ICAR-heuristic will simply traverse all these down-line empty crew assignments (original crew pairings) and mark the flight nodes that it passes. All these will have a crew shortage.

Once the conflicts are identified, these can in principle be resolved in any order. However, only one strategy is applied here, namely that the earliest conflicts are resolved first.

The earliest crew shortage in this example occurs at flight 2. Flight 2 was down-line from flight 12 in the original flight schedule, but flight 12 was cancelled, so no crew units are in effect assigned to flight 2. This is resolved by assigning crew units from the supply node at airport 1. The most suitable crew, with respect to the number of conflicts they resolve, happens to be the crew originally assigned to flight 1, which was cancelled. It is assumed that no hard restriction violations arise due to this assignment. This crew was placed in the supply node when the type 1 conflicts were resolved in step 3. The flight 2 crew shortage and consequent crew reassignment is illustrated in figure 8.17.

Figure 8.17: *Resolving a type 2 conflict: Assigning a real crew to flight 2.*

After DAR in step 2 (section 8.7.2), flight 2 is a part of link 3. As a consequence, when the necessary crew units are assigned to this flight, all the down-line flights in this link are manned to the furthest extent possible without violating hard restrictions. This is due to the fact that the flight link is traversed and conflicts are resolved down-line in accordance with the conflict just resolved.

As can be seen in the original link involving the crew just assigned to flight 2 (figure 8.15), one of the crew units is supposed to end its pairing at airport 3 approximately 300 minutes from the decision point. Likewise, approximately 350 minutes from the decision point, another crew unit is supposed to end its pairing at airport 1. It is assumed that these are hard restrictions, so they must be respected.

Link 3 – as it is after DAR – is illustrated in figure 8.18. Here a real crew

Figure 8.18: *Link 3: After DAR, crew assignment to flight 2 and update of down-line crew to flight assignments.*

has been assigned to crew node 2. In principle, flight 13 and 3 still only have empty crew units assigned to them. However, once a real crew is assigned to flight 2, this crew "flows" through the network in effect "filling out" the empty crew assignments – this is what is referred to as an update.

In figure 8.18, 2 cabin crew units and 1 cockpit crew unit are assigned to flight 2. They are then flown to airport 3 (crew node 12). Originally, a cabin crew unit ended its pairing here while the other two crew units continued along with a new crew unit from the crew supply node. This is *not* changed in the revised solution, even if the crew member ending at airport 3 could have continued. Again, this has to do with retaining as much as possible of the original schedule. This situation appears again at airport 1 in the same figure (crew node 3). Determining which crew unit should continue the pairing in these situations depends on who will resolve the most conflicts further down-line and cause the least delays.

In this example, the crew units from crew node 2 in airport 1 are in fact the crew units that originally undertook flight 13 (see figure 8.15). For that reason this link is completely restored, and deciding which crew units go where becomes a simple matter of respecting the hard restrictions. The only upset is that flight 13 is delayed approximately 50 minutes.

This leaves only two other crew shortages, namely flight 15 and its down-line flight 5. The aircraft assigned to flight 15 also originates from airport 3, so the crew has to be supplied from the crew supply node. The crew units that were assigned to take flight 12 before it was cancelled are an obvious choice. Flight 15 is in flight link 1 updated in the manner described above.

Type 3 and 4 Conflicts

These two conflicts (hard and soft restriction violations and flight delays caused by crew assignments) are very difficult to illustrate in an example. The test instance described in this section does not have any type 3 and 4 conflicts – resolving type 1 and 2 conflicts were sufficient to create an optimized and feasible schedule. A test instance could possibly have been created that contained type 3 and 4 conflicts after resolving type 1 and 2 conflicts. However, to illustrate the workings of the ICAR-heuristic on such a test instance would be extensive and difficult. The general principles are all illustrated earlier and hopefully the idea behind them was made clear then.

8.7.4 Step 5: Repeating the ICAR-heuristic

The final step involves repeating the whole process for different initial solutions. For the purposes of this test instance, this is not relevant. However, note that different initial solutions would be generated by simply making random swaps among the aircraft in the original schedule. This will result in different solutions after DAR as was demonstrated in section 6.9.

8.8 Conclusion on Chapter 8

This chapter describes a method to solve ICARP. The method suggested separates ICARP into two problems, namely an aircraft and a crew recovery problem. DARP is solved first by using the SALS heuristic described and implemented in chapter 6. The crew pairings and assignments are then modified to fit the revised flight schedule. The two problems are not completely separated, i.e. *some* DAR decisions can be made during the crew recovery. Specifically these include delaying flights, rolling flights up or cancelling flights. Only the aircraft to flight assignments cannot be changed during the crew recovery.

The most promising feature of the ICAR-heuristic is its flexibility. Its design is modular and each module is applied in a sequential manner. The first module – the DAR-heuristic – already exists in a version which seems to work well and fast. Further modules can be built to identify and resolve each of the conflicts defined in section 8.5.3 and 8.5.4. Each of these modules can be subdivided into more modules, i.e. different solution methods applied to resolve each conflict. The possibility of subdividing the ICAR-heuristic into modules achieves two important things with respect to implementation: (i) The order in which modules are applied can be interchanged to accommodate different airline priorities, i.e. when should standby crew be applied – as opposed to re-linking – to resolve a crew scheduling problem? (ii) New features in the ICAR-heuristic could be added as modules without changing the existing heuristic structure.

There are of course a number of uncertainties associated with the ICAR-heuristic. Firstly, it is not fully *integrated* because aircraft and crew are considered separately. This may ultimately exclude the possibility of finding optimal solutions to ICARP. Similarly, it may be difficult to find suitable crew pairings/rosters if DARP is solved first without taking any crew considerations into account. Only empirical evidence can show whether or not this is the case.

Chapter 9

Main Conclusion

The purpose of this thesis was to:

1. find and describe a solution method to solve the Dedicated Aircraft Recovery Problem (DARP).

2. implement and test this solution method.

3. describe a solution method to solve the Integrated Crew and Aircraft Recovery Problem (ICARP) using the results from solving DARP.

With respect to the first goal, several methods were explored. A thorough examination of literature on disruption management revealed that a number of researchers have developed methods that solve DARP. However, there were problems with most of these, e.g. the models were too complicated to solve or the decision possibilities were unrealistic. One model developed by Cao and Kanafani (1997) did seem promising. However, an attempt to verify this demonstrated that several errors existed which could not be corrected effectively. In the end, none of the models found in the literature seemed to form a good base on which to make further developments. Instead, a new way of representing DARP was developed (see section 6.3.1) along with a heuristic solution method (DAR-heuristic).

With respect to the second goal, 25 problem instances were generated to test the DAR-heuristic. These consisted of flight schedules of different sizes (50 – 800 flights) with 20% of all flights being delayed for different lengths of time. Various versions of the DAR-heuristic were then applied to each of the problem instances in order to find revised flight schedules of a better quality. The fastest DAR-heuristic – a steepest ascent local search (SALS) – was able to find revised flight schedules in less than 5 seconds on average that were significantly improved. Other versions of the DAR-heuristic did find marginally better revised flight schedules, but given the speed of SALS, it was not enough to justify using those instead. Furthermore, a solution space analysis in the form of fitness landscapes demonstrated that the structure of the solution space is very well-suited for SALS. A number of extensions of the DAR-heuristic were also discussed. Although they were not implemented, it appears that these extension are relatively easy to implement.

With respect to the final goal, a detailed outline of an integrated crew and aircraft recovery heuristic (ICAR-heuristic) was given. It was not implemented, but it was made probable that it could be. It was also made probable that the complexity of an ICAR-heuristic is reasonably small, thus allowing a reasonable computational time frame.

List of Important Terms

The terms below are all specific to the airline industry. They are shown in **boldface** when they first appear in the text. Likewise, the page number where each term first appears is listed – possibly along with pages where the term is very important. Terms that have no page reference are only found here in the list of terms.

Aircraft balance (p. 20 and 102): Describes the situation where the number of aircraft actually terminating at a certain airport equals the number of aircraft that should terminate at this airports.

Airport curfew (p. 17 and 114): Most airports are closed for a period of time during the night. All aircraft arrivals and departures must occur outside this curfew period.

Aircraft density (p. 40): The number of aircraft that depart from an airport within a specified time horizon.

Aircraft roster (p. 2): The plan for each aircraft, i.e. the flights it will undertake that day.

Aircraft shortfall (p. 105): A situation where fewer aircraft terminate at an airport than should have (see also **aircraft balance**).

Aircraft to flight assignment (p. 6): In figure 6.1, an arc connecting an aircraft node to a flight node is called an aircraft to flight assignment. It infers that the aircraft will undertake the flight to which it is assigned.

Aircraft turn-around: When aircraft arrive at an airport, a certain amount of time is spent on safety checks, minor maintenance, cleaning, etc. The time it takes to complete these activities and make the aircraft ready for its next flight is known as the turn-around time.

Aircraft type (p. 16 and 116): Each aircraft is characterized by, for example, a seating capacity and license requirements for both cockpit and cabin crew. Hence, aircraft differ from each other (see also **fleet**).

Base station (p. 10): The airport(s) from which crew initiate and end their **crew pairings**. It is typically also the airport where an airline, for example, has its repair facilities.

Crew assignment (p. 10): When an individual crew unit is assigned to flight. A **crew pairing** thus consists of any number of such assignments.

Crew pairing (p. 10): A sequence of **crew assignments** that can be undertaken by a single crew member.

Crew roster (p. 2): A list of the exact crew units assigned to each **crew pairing**.

Crew schedule: All the crew rosters as a whole is referred to as the crew schedule.

Crew shortage (p. 9): When crew required to undertake a flight are not available. This may occur if earlier flights have been cancelled or crew members fall ill.

Crew to flight assignment (p. 9): In figure 8.1, an arc connecting a crew unit to flight is called a crew to flight assignment. It infers that the crew unit in question will undertake the flight to which it is assigned.

Crew turn-around (p. 130): The time a crew unit rests before it is assigned to the next flight.

Crew type (p. 16 and 120): There several types of crew members in a crew, e.g. captain, 1st officer, customer service director, stewards, and stewardesses etc. In this thesis, two types are considered: (i) cockpit crew and (ii) cabin crew.

Crew unit (p. 120): The unit is a model specific term. It denotes that one or more crew members of the same **crew type** are grouped in a team.

Dead-heading (p. 9): Dead-heading means transporting crew to flights at other airports where a crew shortage exists (see also **crew shortage**). It also includes transporting crew members to their **base station** if they end a pairing away from home.

Decision point (p. 60): The point in time when the heuristic is applied to resolve disruptions that have occurred.

Disruption (p. 2): Any situation where one or more activities in one or more of the key resource areas (e.g. crew or aircraft) have deviated from the resource plan. Subsequent activities in the affected lines of work (see note under **flight link**) either cannot start on time – or can start on time, but only after controller intervention.

Ferrying aircraft (p. 14 and 116): When an aircraft is transported from one airport to another without taking any passengers. This option may be used if an aircraft is needed at another airport.

Fleet (p. 116): Denotes a series of aircraft of the same **aircraft type**.

Flight leg (p. 28): In figure 6.1, flight legs are the arcs connecting flight nodes in one airport to aircraft nodes in other airports.

Flight link (p. 13 and 28): A number of **flight legs** are assembled through aircraft to flight assignments to form a schedule for an aircraft. At this stage in the planning process a specific tail is not assigned to the link (see **tail assignment**).

Flying time (p. 69): The time it takes an aircraft to complete a flight leg, i.e. to travel the distance from the origin to the destination airport.

Here-and-back (p. 141): Flights in a **hub-and-spoke** system are arranged as **round-trips**, where one flight is flown from the hub to the destination airport and the other is flown back to the hub again.

Heuristic setting (p. 60): During a working day one or more disruptions may occur. Given these disruptions the heuristic is applied to solve the problem. The **decision point** is set to the current point in time and the time horizon spans the remainder of the day.

Hub-and-spoke system (p. 66 and 107): This system is a special way to arrange the flights in the flight schedule. A few airports are selected as hubs and flights are then flown **here-and-back** from the hub to the destination airports and back to the hub again.

Out-station: The out-station is simply all other airports than the **base station** with regard to a resource.

Ready-time aircraft (p. 31): The point in time when the aircraft is ready to undertake a flight.

Ready-time crew: The point in time when crew units are ready to man a flight.

Rolled up (p. 137): Denotes that an aircraft is delayed at the decision point, but through reassignments, the delay is reduced, thus the aircraft is *rolled up*.

Round-trip: A round-trip is a pair of flights flown from one airport to the destination airport and back again.

Short haul (p. 61): In short-haul operations, all the short distance flights are serviced. A short-haul operation could be all European destinations for a European carrier or the domestic destinations for an American carrier.

Standby crew (p. 9): Crews on standby are crew members in reserve who are not yet assigned to any pairings. These crew members can be assigned to flights that have been delayed or cancelled due to a disruption, thus resolving the problem.

Surplus aircraft (p. 23): Surplus aircraft are aircraft in reserve that can be substituted with aircraft that are disrupted.

Swap (p. 6 and 23): A swap is when a pair of **aircraft to flight assignments** are interchanged, e.g. the original assignments 1–1 and 2–2 are swapped to 1–2 and 2–1.

Tail assignment (p. 5): Denotes that a specific aircraft is assigned to a

flight link, thus yielding an **aircraft roster**.

Turn-around time (p. 10): The time it takes for either an aircraft or a crew to have a turn-around (see **aircraft turn-around** and **crew turn-around** respectively).

Bibliography

Angel, E. and V. Zissimopoulos (1998). On the Quality of Local Search for the Quadratic Assignment Problem. *Discrete Applied Mathematics 82*, 15–25.

Argüello, M. F., J. F. Bard, and G. Yu (1998). Models and Methods for Managing Airline Irregular Operations. In G. Yu (Ed.), *Operations Research in the Airline Industry*. Boston: Kluwer Academic Publishers.

Barr, R. S., B. L. Golden, J. P. Kelly, M. G. C. Resende, and j. W. R. Stewart (1995). Designing and Reporting on Computational Experiments with Heuristic Methods. *Journal of Heuristics 1*, 9–32.

British Airways (2000, February). *Eurofleet Cabin Crew Manual*. British Airways.

Cao, J.-M. and A. Kanafani (1997). Real-Time Decision Support for Integration of Airline Flight Cancellations and Delays Part I & II. *Transportation Planning and Technology 20*, 183–217.

Jarrah, A. I. Z., G. Yu, N. Krishnamurthy, and A. Rakshit (1993). A Decision Support Framwork for Airline Flight Cancellations and Delays. *Transportation Science 27*, 266–280.

Larsen, J., A. Larsen, T. Hultberg, and A. Ross (2001, February). Dedicated Crew Recovery in Descartes. Technical report, Informatics and Mathematical Modelling (IMM). Techical University of Denmark (DTU).

Lettovsky, L. (1997). *Airline Operations Recovery: An Optimization Approach*. Ph. D. thesis, Georgia Institute of Technology, Atlanta, USA.

Mladenović, N. and P. Hansen (1997). Variable Neighborhood Search. *Computers & Operations Research 24*, 1097–1100.

Stojković, M. and F. Soumis (2000a, January). An Optimization Model for the Simultaneous Operational Flight and Pilot Scheduling Problem. Technical Report G-2000-01. Technical report, GERAD and École Polytechnique de Montreál.

Stojković, M. and F. Soumis (2000b, June). The Operational Flight and Multi-Crew Scheduling Problem. Technical Report G-2000-27. Technical report, GERAD and École Polytechnique de Montreál.

Stützle, T. (1999). Iterated Local Search for the Quadratic Assignment Problem. Technical report, Darmstadt Technische Hochschule.

Taillard, E. (1991). Robust Taboo Search for the Quadratic Assignment Problem. *Parallel Computing 17*, 443–455.

Teodorović, D. and S. Guberinić (1984). Airline Optimization. *European Journal of Operational Research 15*, 178–182.

Teodorović, D. and M. Stojković (1990). Model for Operational Daily Airline Scheduling. *Transportation Planning and Technology 14*, 273–285.

Teodorović, D. and M. Stojković (1995). Model to Reduce Airline Schedule Disturbances. *Journal of Transportation Engineering 121*, 324–331.

Yan, S. and D.-H. Yang (1996). A Decision Support Framework for Handling Schedule Pertubations. *Transportation Research 30*, 405–419.

Appendix A

Implementation of the CCD Model in *GAMS*

The CCD model was implemented and tested using *GAMS*. The model is named: **airmodel.gms**, and can be found in the directory:

```
/home4/proj/proj48/Public/CCD_model/
```

A *C++*-program was used to generate most of the parameters used in the CCD model. The data file containing these parameters are found in the same directory named: **problem.dat**.

When the model is solved a *GAMS* file writes the solution in a data file. The former is named: **solution.gms**, and the latter is named: **solution.dat**.

The *C++*-program itself is also found in the directory. Below is the main file listed along with the header files used (all the standard header files are not mentioned):

first_schedule.cpp uses the following header files:

- **first_schedule.h** – declares a few global variables.
- **dtypes.h** – declares all the data types used.
- **init_var.h** – initiate all the variables.
- **make_data.h** – generates the data.
- **output.h** – writes the file **problem.dat** among other things.

- **input.h** – reads the file **solution.dat** to interpret the solution found by *GAMS*.

The header files only contain the external function declarations (except the file **dtypes.h**) and the respective source code files are found in the same directory changing the extension ".h" to ".cpp".

Finally, a file is used to compile and link all object files (i.e. the files with the extension ".o"). This file is named: **makefile**.

The *GAMS* model was implemented using approximately 300 lines of code, while the *C++*-program contained more than 800 lines of code.

Appendix B

Implementation of the Iterated Local Search (ILS) in *C++*

The Iterated Local Search (ILS) heuristic was implemented using more than 2000 lines of *C++*-code, and the source code is found in the following directory:

```
/home4/proj/proj48/Public/ILS_heuristic/
```

Below is the main file listed along with the header files used (all the standard header files are not mentioned):

ils.cpp uses the following header files:

- **dtypes.h** – declares some of the data structures used. Here all constants are also defined.
- **links.h** – declares a class which contains the flight links of the original schedule.
- **stations.h** – declares a class which contains data on the different flights in the schedule.
- **solution.h** – declares a class which contains the solution data structure.

The header files contain the declarations of the classes (i.e. their attributes and methods), except the file **dtypes.h**, and the respective source code files are found in the same directory changing the extension ".h" to ".cpp".

Finally, a file is used to compile and link all object files (i.e. the files with the extension ".o"). This file is named: **makefile**.

Appendix C

Experimental Results

The following experiments has been conducted with the standard and revised versions of the ILS heuristic. All the experimental results can be found in the following directory:

```
/home4/proj/proj48/Public/Experimental_results/
```

C.1 Results of Experiments with the Standard ILS Heuristic

The following 13 files contain the tables with the results of the runs with the ILS heuristic for 3 minutes.

- **ILS_table1.ps** – contains the test instances 1 and 2.
- **ILS_table2.ps** – contains the test instances 3 and 4.
- **ILS_table3.ps** – contains the test instances 5 and 6.
- **ILS_table4.ps** – contains the test instances 7 and 8.
- **ILS_table5.ps** – contains the test instances 9 and 10.
- **ILS_table6.ps** – contains the test instances 11 and 12.
- **ILS_table7.ps** – contains the test instances 13 and 14.
- **ILS_table8.ps** – contains the test instances 15 and 16.
- **ILS_table9.ps** – contains the test instances 17 and 18.
- **ILS_table10.ps** – contains the test instances 19 and 20.

- **ILS_table11.ps** – contains the test instances 21 and 22.
- **ILS_table12.ps** – contains the test instances 23 and 24.
- **ILS_table13.ps** – contains the test instance 25.

C.2 Results of Experiments with the ILS Heuristic with a Duration of 24 Hours

A text file is produced that contains the results of the runs with the ILS heuristic for 24 hours. This file is named: **24hour_test_results.txt**.

C.3 Results of Experiments with the Revised ILS Heuristic

The following 13 files contain the tables with the results of the runs with the RILS heuristic for 3 minutes.

- **RILS_table1.ps** – contains the test instances 1 and 2.
- **RILS_table2.ps** – contains the test instances 3 and 4.
- **RILS_table3.ps** – contains the test instances 5 and 6.
- **RILS_table4.ps** – contains the test instances 7 and 8.
- **RILS_table5.ps** – contains the test instances 9 and 10.
- **RILS_table6.ps** – contains the test instances 11 and 12.
- **RILS_table7.ps** – contains the test instances 13 and 14.
- **RILS_table8.ps** – contains the test instances 15 and 16.
- **RILS_table9.ps** – contains the test instances 17 and 18.
- **RILS_table10.ps** – contains the test instances 19 and 20.
- **RILS_table11.ps** – contains the test instances 21 and 22.
- **RILS_table12.ps** – contains the test instances 23 and 24.
- **RILS_table13.ps** – contains the test instance 25.

C.4 Results of Experiments with the Steepest Ascent Local Search Heuristic

A text file is produced that contains the results of the experiments with the Steepest Ascent Local Search (SALS) heuristic. The file is named:

SALS_test_results.txt.

C.5 Results of Experiments with the Repeated SALS Heuristic

A text file is also produced that contains the results of the experiments with the Repeated SALS heuristic. This file is named: **RSALS_2000_overview.txt**.