Airline Crew Scheduling During Tracking

Jesper Holm

LYNGBY 2002
MASTER THESIS
NR. ??/02

IMM

Trykt af IMM, DTU

Preface

This M.Sc. thesis is the final requirement for obtaining the degree: *Master of Science in Engineering*. The work has been carried out in the period from 1st of September 2001 to 28th of February 2002 at the Operations Research section at Informatics and Mathematical Modeling, Technical University of Denmark. The work has been supervised by Professor Jens Clausen.

The thesis has been carried out in collaboration with Scandinavian IT Group where Kim Milvang-Jensen and Steven Laursen have been mu collaborators.

Lyngby, February 28th, 2002

Jesper Holm
c961050

Abstract

This thesis investigates airline crew scheduling, which covers the problem of assigning crew to planned departures. The problem is often subdivided into several phases: The pairing phase, the assignment phase, the tracking phase and the day-to-day phase. In this thesis, the tracking phase has been the main focus. An existing system developed by Scandinavian IT Group has been used as a starting point for the developed software.

The properties of the tracking phase make it desirable to construct individual work shifts for each crew. This approach is in contrast to the usual approaches taken during the assignment and pairing phases. Hence, a set of individual work shifts is constructed for each crew to be considered. From each set, a work shift for each crew to man is chosen using a Set Covering formulation.

The construction of work shifts is done by enumerating a subset of all possible work shifts by using heuristics or by using an existing generator developed by Scandinavian IT Group. The resulting Set Covering Problem is solved using Simulated Annealing.

A total of 16 "real-life" problems are considered. The smallest contains slightly more than 200 crew and 150 open flights and the largest contains just above 600 crew and 900 open flights. Using the developed heuristics around 90% of the planned departures were covered. In comparison, the existing heuristic resulted in a coverage around 60%.

Keywords Airline crew scheduling, Simulated Annealing, Combinatorial Optimization, Heuristics.

Contents

Chapter 1

Introduction

Airline crew scheduling is the problem of assigning personnel (crew) to planned departures. Because crew cost make up one of the largest direct expenses when operating an airline company, optimizing crew utilization may result in huge gains. Thus the area of crew scheduling is important.

Unfortunately, obtaining good solutions to the airline crew scheduling problem is hard. First of all because the problem is combinatorial in nature; there are a huge number of ways one can man the hundreds of departures an airline company serve per day. However, the problem is not restricted to a single day and often everything between a couple of days and up to a month has to be considered. On top of this a large set of rules given by the aviation authorities and union agreements has to be respected. Just checking the legality of a given solution is difficult and computationally expensive.

This project has been carried out in collaboration with Scandinavian Airline Systems (SAS). For further details see section 2.1.

Below an introduction to the terminology used in the field of airline crew scheduling will be presented. Followed by a more in-depth introduction to airline crew scheduling. Finally an overview of the rest of the report is given.

1.1 Terminology

In order to be able to give a more specific problem formulation an introduc
tion to some of the specialized terminology used in airline crew schedulin
will be presented below.

Crew A crew is a single person. There are two main types of crew: Cabi
(servicing the passengers) and cockpit (steering the aircraft). In th
project only cabin crew are considered and hence crew will be use
as an synonym for cabin crew. However the ideas and method use
should be easy to apply to cockpit crew as well.

Base A base is an airport. A *homebase* is the base where a crew "belongs
For SAS this is one of the following: Copenhagen (CPH), Stockholm
(ARN), and Oslo (OSL).

Connection time The period of time between a crew arrives at a bas
with one flight and depart with another.

Pairing A pairing is a work shift for a single crew. On figure 1.1 th
structure of a pairing is shown. A pairing starts and ends on th
same base (on the figure CPH). A pairing is constructed from one o
more *duty periods* which are series of flights (also known as *legs*) wit
a small connection time. The period of time between two legs in
duty period is called a *sit*. Each duty period starts with a briefing an
terminates with a debriefing. When the connection time between tw
legs is large it is called a *stop*, which separates duty periods. Pairing
might also be referred to as *rosters* or *slings* (the latter only used b
SAS).

Open flight Open flight is used to denote several slightly different things
First of all *an* open flight is a flight which lacks one or more crew
But *one* open flight is also used to denote that one crew is missin
on a flight. This means that if a flight is lacking two crew this fligh
represents two open flights; one for each crew.

Passive transfer is also known as *deadheading*. A crew is said to be on
passive transfer when she is assigned to a flight that is not in lack o
crew. This is done to transport her from one base to another, eithe
because she is needed at the arriving base or to return her to he
homebase.

Standby also known as *reserve crew* is a crew that can be called on wor
with a relative short time of notice (often a crew has to be at he
homebase within an hour).

Legality A pairing is said to be legal if it complies with the set of rules given by the *aviation authorities* as well as *union agreements*. This set has to be satisfied by all pairings.

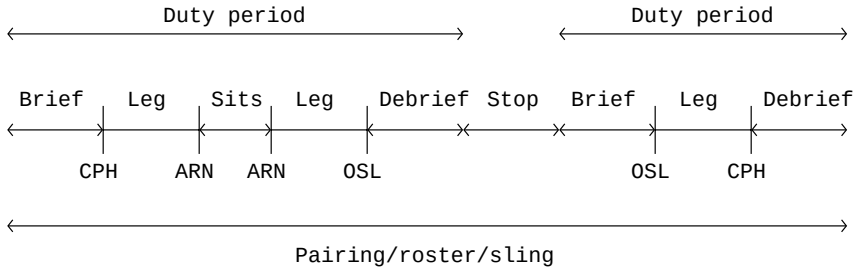


Figure 1.1: *The building blocks of a pairing.*

1.2 Phases of airline crew scheduling

Due to the difficulties of airline crew scheduling the solution is traditionally divided into a number of phases. On figure 1.2 these phases are sketched. The first phase consist of longterm *planning* where the destinations to serve and the type of service (number of flights per day) are determined. This results in the set of all flights to fly.

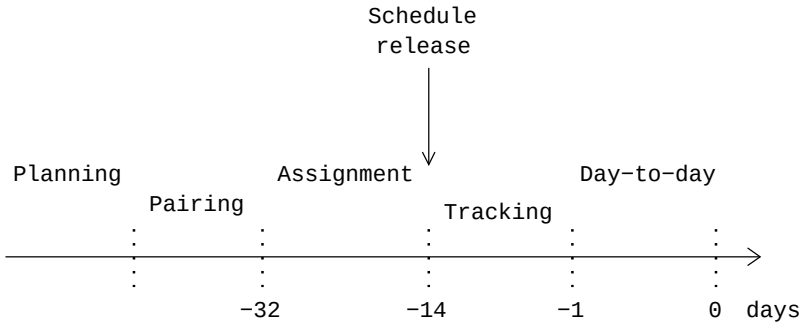


Figure 1.2: *Phases of airline crew scheduling.*

The aim of the next phase, the *pairing*, is to generate pairings from th planned flights from the planning phase. The objective is to include eac planned flight exactly once in the generated pairings.

When pairings have been constructed individual crews are assigned to th generated pairings in the *assignment* phase. This step ends with a schedul which gives the relation between crews and flights; showing the flights an the crews that are to man them. At this point the schedule is released, s each crew gets to see the flights she is to man.

The released schedule is not complete, as not all flights are assigned th number of crews which are needed. In other words the released schedul most likely contains open flights. Some of the open flights in the schedul are inherited from the previous phase. Others emerges during the trackin phase due to new passenger forecasts[1] and illness. The objective of th *tracking* phase is to assign crew to these open flights.

Finally, on the day of operation, the schedule is executed in the *day-to-da* phase. Remaining open flights, new which arise, delays etc. are dealt wit possibly by cancellation.

1.3 Report organization

Problem description This chapter gives a more precise description o the problem targeted in the project along with a description of th project background.

Tracking phase at SAS This chapter describes an existing approach fo solving the problems of airline crew scheduling during the trackin phase, used by SAS.

Solution approach This chapter describes the approach taken in th project for solving the problem.

Implementation Here an overview of the implementation is given.

Results This chapter presents the numerical result obtained along wit a comparison with results obtained with the program currently use by SAS.

[1] The number of crew needed on a flight is determined by the type of flight *and* th estimated number of passengers.

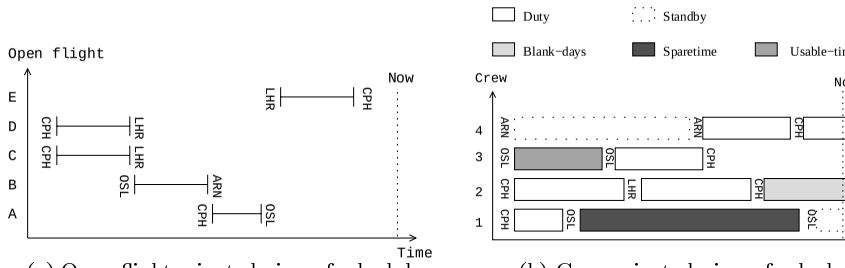
Chapter 2

Problem description

The problem considered in this project is crew scheduling during the tracking phase. As described in section 1.2 a schedule has already been constructed when entering the tracking phase. However, open flights are still around during the tracking phase and the problem is to assign crew to these flights. To be able to solve this problem there has to be crew available that can be assigned to the open flights. Therefore, different kinds of standby time are allocated in the schedule during the assignment phase, which can then be used during the tracking and day-to-day phases.

A graphical presentation of the problem is given on figure 2.1. On figure 2.1(a) an open flight oriented view of the schedule is given – simply showing the open flights. Notice that open flights `C` and `D` originates from the same "physical" flight between Copenhagen and London which lacks two crew members. This shows up in the schedule as two different open flights.

On figure 2.1(b) the schedule is shown from a crew point of view which for each crew gives the different *activities* that she has been assigned. *Duty* time is time where a crew has already been assigned to flights; she is working. *Standby* is time where crew can be called on work with a relatively short notice (often within an hour). Other types of standby are *blank-days* and *usable-time*. The differences lie in the specific rules for how and when they can be used to close open flights. *Sparetime* is the time where crew is off work and therefore cannot work. Finally there are "holes" in schedule where crew is not assigned to a specific activity. Crew 3 has a hole in her



(a) Open flight oriented view of schedule (b) Crew oriented view of schedu

Figure 2.1: A sample schedule showing open flights (left) and crew schedul (right).

schedule after returning to Copenhagen from Oslo. Holes *might* be use when closing open flights.

All in one, standby, blank-days, usable-time and holes in schedule are re ferred to as resource periods; time in schedule where crew might be use to close open flights. This gives the following problem statement:

> Close as many open flights as possible using the resource periods allocated in the schedule as cheaply as possible, with respect to some cost measure.

On top of this a *crew oriented* approach should be tested when solving th problem. This means that information about the crew that is to man pairing should be sought used when the pairing is generated. This is i contrast to the approach described above where pairings are generated i the pairing phase and assigned in the assignment phase.

2.1 Project background

The project has been made in a collaboration between Informatics an Mathematical Modelling (IMM), Technical University of Denmark and Scan dinavian IT Group (SIG) the latter being 100% owned by Scandinavia Airline System (SAS).

SIG has developed a system called TAP-AI that prior to 1998 was successfull used to solve the scheduling problem during the tracking phase. Howeve

due to the introduction of new union rules the heuristics deployed by TAP-AI has become invalid. From 1998 and onwards the tracking phase has therefore been dealt with manually. However, the combinatorial nature of the problem indicates that an OR-approach might lead to notable savings.

TAP-AI has a pairing generation part that has been sought used to generate pairings of open flights which then could be used by the staff. However, since no information about the available resources in schedule is used it is not guaranteed that the generated pairings can be assigned to a crew. Practices shows that the slings generated by TAP-AI often do not fit with the resources in schedule and TAP-AI is not currently used by the staff.

Chapter 3

Literature review

Currently, there has been much work carried out targeting the pairing assignment and day-to-day phases. But almost no (published) work ha been done directly targeting the tracking phase. It has only been possible t find one paper [13] which deals directly with this phase. But the technique and ideas used in the pairing, assignment and day-to-day phases are a usable when considering scheduling during tracking.

The literature can roughly be divided into 3 classes: Those that deal wit the pairing and assignment phases, the one that deals with tracking an those that try to solve day-to-day problems.

3.1 Pairing and assignment

These problems have been targeted in two main ways; as two separate problems and as one. In both cases a variation of the Set Partitionin Problem is used to perform some kind of selection among pairings:

$$\text{Minimize} \quad Z = \sum_{j=1}^{n} c_j x_j \tag{3.1}$$

Subject to

$$\sum_{j=1}^{n} a_{ij} x_j = 1, \quad \text{for } i = 1, \ldots, m \tag{3.2}$$

$$x_j \text{ integer}, \quad \text{for } j = 1, \ldots, n \tag{3.3}$$

a_{ij} arises from a matrix A where each row corresponds to a flight and each column to a pairing. x_j denotes if pairing (column) j is chosen, the corresponding cost is c_j. (3.2) ensures that each flight is included exactly once in the set of chosen pairings. Alternatively, a set covering model might be chosen where the equal sign in (3.2) is replaced by a greater-than-or-equal sign. Then each flight is covered at least once by the chosen pairings. Both models are known to be NP-hard. However, a feasible solution to the Set Covering Problem can easily be found given it exist: Just include all columns in the solution. This approach can obviously not be used when dealing with the Set Partitioning Problem because of over coverage of rows is not allowed. This makes the Set Covering Problem much nicer to work with. However, since over coverage of rows is allowed one might get multiple coverage of flights in the Set Covering Problem formulation.

Another common element is a graph G which represents possible pairings. Each path in the graph represents a (legal) pairing which then again corresponds to a column in A. Some kind of cost or restriction is often introduced in G.

In [8] a genetic or evolutionary algorithm[12] is developed that solves the Set Partitioning Problem arising from crew scheduling as described above. Neither pairing construction nor the assignment of pairings to crew are considered.

In [14] instead of using individual flight legs as building blocks, a set of duty periods (see figure 1.1) is constructed. These duty periods are then selected in a Set Covering Problem so that a set of duty periods covering the flight legs is obtained. The selected duty periods are organized in a graph where paths correspond to pairings. The problem is solved by column generation where a shortest path algorithm is used on the graph to generate columns

to a master problem. This again is a Set Partitioning Problem with row representing the selected duty periods and columns representing pairing A very simple form of legality is sought enforced on the generated pairing through the graph representation. The initial construction of duty period is not considered, nor is the assignment to crew.

In [4, 6] a graph is constructed directly from the flight legs and used t enumerate possible pairings. In [6] the pairings are constructed once an then a Set Partitioning Problem is solved. In [4] a graph G^k is constructe for each crew k and a path in G^k now represents a legal pairing for crew k A shortest path problem on these graphs are used as the subproblem in column generation scheme. The master problem – again – becomes a Se Partitioning Problem. Because a pairing always has a crew associated th assignment problem is also solved. G^k is constructed by enumerating a possible and legal pairings for crew k.

[3] formulates a mathematical model that given a set of pairing P^k fo each crew k selects exactly one pairing from each set P^k so that all flight are covered exactly once. This model is solved using a branch-and-boun technique.

In [9] a Simulated Annealing approach is taken to solve the assignmer problem (it is assumed that a set of pairings is given). Firstly, an initia assignment of crew to pairings is made using some heuristic. The neigh bourhood is defined by either moving one pairing from one crew to anothe or by swapping two pairings between two crew.

3.2 Tracking

As mentioned above, the only paper found dealing directly with the trackin phase is [13]. Firstly, it separates the introduced phases of airline cre scheduling into two meta phases; a planning phase and an operationa phase. This is outlined on figure 3.1.

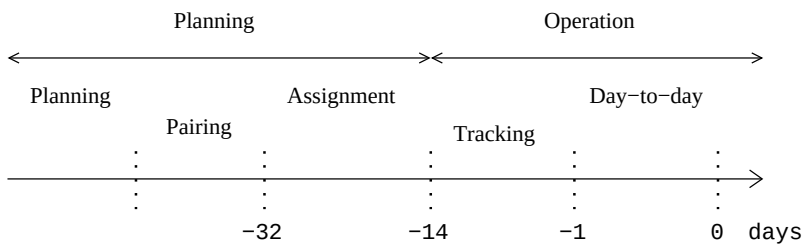


Figure 3.1: *The division of phases of crew scheduling into to meta phases: The planning phase and the operational phase.*

[13] defines the *operational airline crew scheduling problem* as that of "modifying individual monthly work schedules for airline crew members during the operational phase of the planned schedule".

The approach taken is column generation. The master problem is modelled as a Set Partitioning Problem over the open flights. The subproblem generates columns corresponding to pairings. The subproblem is formulated as a shortest path problem on a *duty graph* G^k for each crew k. Each path in the graph corresponds to a legal pairing. The cost associated with each path in the G^k corresponds to the marginal cost of the new pairing in the master problem.

The specific problem considered in this paper was that of cockpit crew.

3.3 Day-to-day

The day-to-day problem is also referred to as airline irregular operations [15] or crew recovery [7], because it consists of targeting those problem that arise due to disruption from maintenance problems, bad weather conditions etc.

In [15] depth first search in a branch-and-bound tree is used to assign crew to flights that have become open due to disruption. The branch is done on the assignment of a crew to an open flight.

In [7] a graph G^k representing pairings is build for each crew k as an extension of the already flown pairings by k. This is then used in a set

covering formulation which ensures that each open flight is covered at leas
once.

Chapter 4

Tracking phase at SAS

This chapter describes the solution approach used currently by SAS when dealing with the tracking phase.

SAS uses a program called TAP-AI, based on self developed heuristics (which will be described below) to close open flights during the tracking phase.

Before 1998 TAP-AI was able to efficiently close open flights. This was done using reordering (from Danish "omdisponering"), which basically swaps pairings in and out of the schedule, possibly changing already planned duty (a more detailed description follows). But in 1998 a new union agreement heavily limited the possibilities of changing planned duty time during the tracking phase. Hence, the usefulness of this approach was limited.

TAP-AI consists of two main parts: One that does the reordering, and one, that generates pairings from the initial set of open flights. To overcome the difficulties of the new union agreement SIG took the approach of letting TAP-AI generate pairings which could then be assigned to crew using allocated resources in schedule (such as standby, blank-days, usable-time and "holes") instead of changing already planned duty – which was the case when using reordering. Thus mirroring the pairing and assignment phases from figure 1.2. The assignment is done manually.

In the following a more in-depth description of TAP-AI will be given.

4.1 TAP-AI

The pairing generation part approach of TAP-AI can be divided into tw steps; an initial pairing generation, which was the generator used befor 1998, and the (main) pairing generation. The latter which has been buil on top of the initial pairing generation after 1998.

Both the pairing generation and the reordering parts of TAP-AI uses th initial pairing generation to build an initial set of pairings from the give set of open flights. Firstly, a description of this initial pairing is given Secondly, a closer look will be taken at the reordering approach used prior t 1998, and thirdly, the main pairing generation used to day will be describe

4.1.1 Initial pairing

The initial pairing uses two lists, which are sketched on figure 4.1 All ope flights are kept in a list `L1`, which is sorted accordingly to increasing de parture time. `L1` is traversed several times with different heuristics tryin to construct pairings. All the heuristics are greedy in the sense that the start at the head of `L1` and add the first flight to the pairing currentl under construction, which fulfill some requirements given by the heuristi Each time a pairing is successfully constructed it is stored in `L2` and th flights which make up the pairing are marked as used in `L1` (illustrate with crosses on the figure). As several heuristics will be tried in turn onl flights *not* marked as used will be considered when trying to construct pairing.

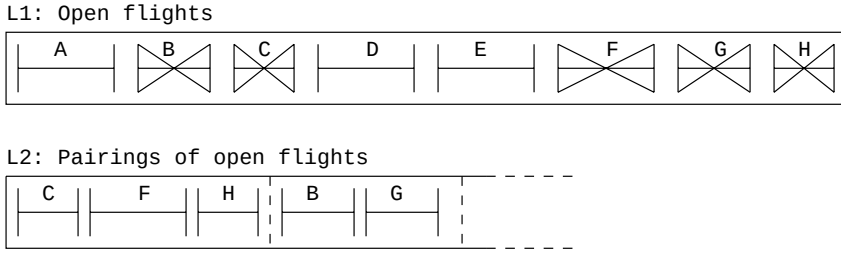


Figure 4.1: *The 2 lists used when* TAP-AI *does the initial pairing on the open flights.*

As described, the initial pairing consist of a number of heuristics. They can be divided into two main categories: A preprocessing (enforcing some rules) and a number of heuristics which constructs pairings. The two categories will be described next.

Preprocessing

First, a small amount of preprocessing is applied. This forces flights to be connected and locked[1] if the "50 minute" rule[2] apply. Similar the "trivsel stop" rule[3] is checked, and flights that are required to be connected will be connected and locked.

Heuristics

Next the following heuristics, which tries to build pairings from the flights in `L1` are applied. The heuristics are tried successively in the order they are listed.

The first heuristic tries to construct a pairing starting from a homebase `X`, with one or more stops with good connection time (less than 3 hours) at non-homebases (here `a` and `b`) and returning to `X`. This is illustrated on figure 4.2.

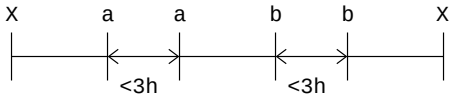


Figure 4.2: *Pairing with good connection time ending and starting at a homebase `X`.*

The second heuristic relaxes the requirement on the connection time; now connection times up to 30 hours for the leg returning to the homebase

[1] Pairings can be locked so they will not be changed at any point in the future.

[2] The rule states that if the captain or the pilot, for a given leg, have less than 50 minutes connection time to the next leg, they should be captain and pilot on that next leg.

[3] Trivsel stop rule forces flights together in pairings that will keep crew "happy".

are accepted. Hence, the requirement becomes $< 30h$ at figure 4.2 for th connection time at base `b`.

The third heuristic tries to build a pairing consisting of exactly two flight with a good connection time and start and end at the same homebase This is illustrated at figure 4.3.

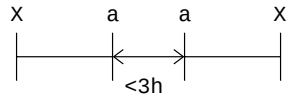
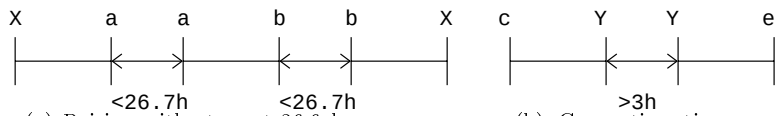


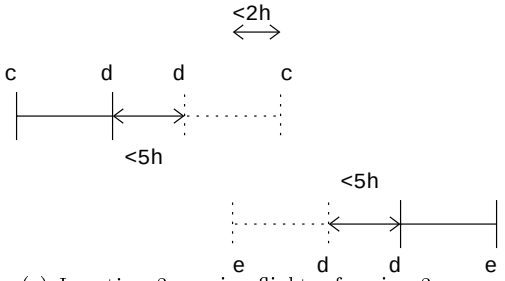
Figure 4.3: *Pairing with exactly 2 legs, good connection time and start and end at a homebase `X`.*

The fourth heuristic relaxes the requirement of good connection time, henc a connection time of up to 26.6 hours is accepted (see figure 4.4(a)). How ever, there are two exceptions illustrated in figure 4.4(b) and figure 4.4(c A flight creating more than 3 hours connection time on a homebase `Y` is no accepted, if the previous base `c` or the next base `e` are a homebases (illus trated at figure 4.4(b)). A long stop (more than 3 hours) is not accepte on a base `d` (see figure 4.4(c)) if it is possible to insert two passive transfe (dotted lines) forming two new balanced pairings. The two passive trans fers may only have 5 hours of connection time and should not overlap wit more than 2 hours.



(a) Pairing with at most 26.6 hours connection time with start and end at a homebase X.

(b) Connection time > 3 hours on homebase Y and c or e also homebase.



(c) Inserting 2 passive flights, forming 2 new pairings, with at most 5 hours connection time and no more than 2 hours of overlap.

Figure 4.4: *Relaxation on connection time (a), with 2 exceptions (b) and (c).*

The fifth heuristic relaxes the requirement from figure 4.4 where the pairing should start and end at the same homebase. Hence, the pairing should start and end on a homebase, but not necessarily the same.

The sixth heuristic is a variant of figure 4.2. It tries to connect a flight, which starts on a homebase X, with one or more flights with good connection time. But now an ending on a base (even a non-homebase) different from the stating base X is accepted.

The seventh heuristic tries to build pairings starting from any base, with good connection time, ending on a homebase (see figure 4.5). A possible bad connection time of up to 24 hours is accepted on the leg arriving on the homebase, with the exception illustrated on figure 4.4(c).

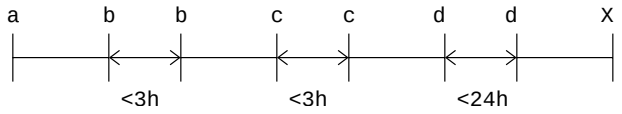


Figure 4.5: *Pairing starting from any base with good connection time and ending on a homebase with possible bad connection time.*

The eighth heuristic tries to construct a pairing starting on a homebas with three or more flights with good connection time (< 3 hours) and endin on any base.

The ninth heuristic tries to construct pairings consisting of two flights star ing at any base a, with good connection time, and ending at base a (se figure 4.6).

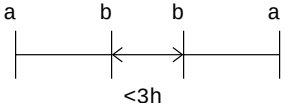


Figure 4.6: *Pairing starting and ending at base a with good connection time at b.*

As a final processing all remaining unmarked legs in L1 from figure 4.1 a converted into pairings consisting of one flight, and are inserted into L2.

4.1.2 Reordering

The reordering approach was developed based on interviews with the sta who prior to TAP-AI solved the scheduling problems during the trackin phase manually. The goal was to develop a system which mirrored th heuristics that the staff consciously or unconsciously would use. An i depth description of reordering is given in [11]. Below a rough outline the ideas used in reordering will be given.

Reordering is a process where pairings are swapped in and out of cre schedule using some heuristics. The process can be described as a loc

search heuristics that tries to transform a current schedule by swapping pairings (of open flights) into the schedule (possible) replacing other pairings which then become open flights. The hope is, that if the pairings that are swapped into the schedule, are always "harder to cover" than the ones they are replacing, the set of open pairings can, at some point in time, be inserted into the schedule without replacing any other pairings.

To be able to use the above sketched local search a definition of "hard to cover" and "easy to cover" pairings/flights has to be introduced. The following characteristics are considered:

Length Several short pairings are considered easier to cover than one long. The hope is that small pairings can be more easily fitted into the schedule than long ones.

Destination Flights between the three homebases are considered easy to cover. This is natural since a huge number of personal are transported with passive flights between the homebases and they can be used to cover flights.

Departure time Pairings which are closer to operation are considered more important to cover than pairings which are far from operation.

If it is not possible to get all open pairings closed, the set of remaining open pairings (after the reordering process is terminated) should hopefully be easier to cover than the initial set. Due to the last preference listed (departure time) some important time has been achieved because open flights are moved forward in time.

4.1.3 Main pairing

The aim of the main pairing (in the following just referred to as pairing) process is to produce pairings which can be covered with the resources allocated in crew schedules. The pairings produced by the initial pairing are optimized towards reordering of crew schedules, because it was originally build to produce the initial set of pairings from the set of open flights used in the reordering part of TAP-AI (as described above). Pairings used in the reordering step are (in general) shorter than the available resources in crew schedules. Therefore some further pairing is introduced to optimize the use of resources.

The pairing process can run in one of two modes. In mode 1 one open flight is only included in one pairing (similar to the way the initial pairing works,

by marking open flights as used). This way one can be sure, that a flight not overcovered, because multiply instances of the same open flight is no present in several pairings.

Mode 2 does not mark flight as used thereby producing several pairing possible containing the same open flight. This is dealt with by a greed heuristics that postprocesses the set of generated pairings and selects subset that do not contain any duplicate use of flights.

The two different modes reflects two different (and greedy) ways of dealin with the set partitioning problem that lies beneath; generate pairings tha covers each open flight exactly once.

In the following descriptions of the heuristics used in the two modes ar given.

Mode 1

Similar to the initial pairing a number of heuristics are tried in turn. The all function on a list of pairings sorted by departure time (see figure 4.7 The heuristics start at the head of the list, and try to build a new pairin constructed from pairings from the list. When a new pairing `X` is built it appended to the list and the pairing used (`B`, `E` and `G`) are marked as use and ignored afterwards.

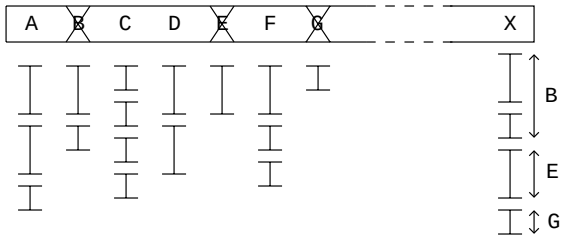


Figure 4.7: *List of pairings.*

The first heuristic traverses the list of pairings looking for pairings, whic starts or ends on a homebase. When such a pairing is found it is checked it starts on a homebase. If it does not start on a homebase passive transfe

is added (if possible) to make it start on a homebase (making sure the pairing is still legal). If this succeeds the result is a pairing like `Pairing 1` on figure 4.8, where `X` is a homebase, the leg `X-b` is possibly passive and `Y` is a homebase. Next the list of pairings is traversed looking for another pairing `Pairing 2` with a connection time less than 5 hours if `X=Y` or less than 15 hours if `X≠Y`. `Pairing 2` is appended to `Pairing 1` if the new pairing is legal, and if one of the following holds:

- The new pairing is balanced.
- If it starts and ends on a homebase and covers at most two days.
- If it covers at most two days and can be balanced by adding passive transfer.

New pairings are appended for as long as possible, under the rules described above along with the 5/15 hours rule, forming one long pairing.

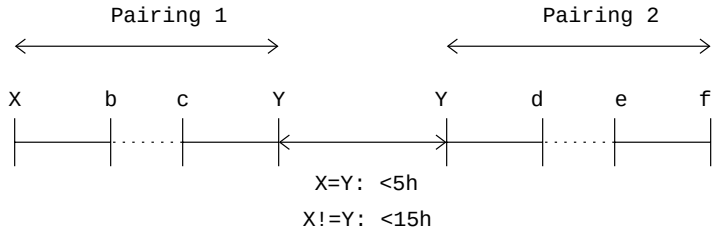


Figure 4.8: *`Pairing 2` are appended at `Pairing 1` by the 5/15 hour rule.*

The second heuristic is just like the first with the exception that passive flights are considered (when trying to connect the two pairings) if there is at most 24 hours between them and if it does not create a night stop on the base where the pairing starts.

The third heuristic tries to balanced pairings, which either starts or ends on a homebase, by adding passive transfers. Optionally it is also attempted to remove the first or the last leg if this would balance the pairing.

The fourth heuristic tries to connect unbalanced pairings that starts and ends on a homebase (possible with a passive transfer) to make them balanced.

Finally, all pairings that contain a night stop on the starting base are broken into smaller pairings on those bases.

Mode 2

Mode 2 is very similar to mode 1 except that it does not mark pairing as used. This is illustrated on figure 4.9, which corresponds to figure 4.7 Hence all possible combinations of pairings starting with `B`, that fulfills th requirements of the given heuristic, are built.

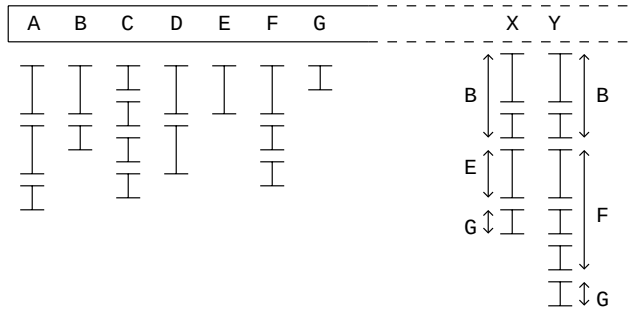


Figure 4.9: *List of pairing.*

Mode 2 therefore produces pairings, where several instances of the sam open flight may be present in, several different pairings. As describe above, the problem now is to solve a set partitioning problem. This is don using a greedy heuristic which sorts the pairings by density. The densit of a pairing is defined as the total number of flights in the pairing minu the number of passive transfers. Hence, the density is a pairing qualit measure. The heuristic chooses the pairing with the highest density firs and rule out all other pairings that contains flights also present in th chosen pairing. This way a best first principle is used.

The final result from the pairing process (despite the mode) is a set o pairings covering each flight exactly once.

However, mode 2 is not considered fully developed and has therefore no been put into production.

4.2 The pros and cons of TAP-AI

As already mentioned, the usefulness of the reordering part of TAP-AI has been heavily limited by the new union agreements.

Out of the two modes that does pairing generation, only mode 1 has made it into production. The problem with mode 2 is that it uses a huge amount of memory and is slow compared with mode 1. Mode 1 is able to quickly produce a set of pairings which covers the open flights. However, since no information about the available resources in schedule are used, there is no guarantee that the generated pairings can be assigned to crew. In practice it has turned out, that the pairings generated by mode 1 often do not fit the available resources in schedule. Thus mode 1 is rarely used by the staff at SAS.

Another drawback are the heuristics used in the initial pairing. As one might have noticed they are redundant and still reside in PROLOG, where the rest of TAP-AI is implemented in C/C++. Hence, maintenance of the initial pairing is harder and making TAP-AI more complex as a whole more complex.

Chapter 5

Solution Approach

As already described the new union agreements have made TAP-AI effe
tively useless. The attempt made to overcome these new restrictions was t
make TAP-AI a pure generator which could be used by the staff in the track
ing department. However, as already described it has not been successfu
due to the quality of the generated pairings. Therefore SIG has looked for
way to improve the generated pairings and, if possible, a way of assignin
crew to the generated pairings as well.

The usual approach for pairing construction does not include informatio
about the crew that (at some point in time) is going to fly the pairing
This makes sense in the pairing phase (see figure 1.2) because the availab
resources in the assignment phase are quite uniform among crew; no c
little work has been assigned to crew at this point. However, in the trackin
phase resources are spread more non-uniformly among crew because a larg
number of pairings already have been assigned to crew. Therefore th
pairings that are going to be assigned in the tracking phase has to b
tailored to the crew that has to cover them. Firstly, the crew has to b
available in the period of time the pairing covers. Hence she is going to b
on some form of standby (standby, blank-day, usable-time etc.). Secondl
the start and end bases of the pairing have to fit with the base at which th
crew is standby or it has to be possible to use passive transfers to transpo
the crew to/from the start/end of the pairing. Thirdly, the pairings hav
to be legal.

The main idea which SIG had considered was this tailoring of pairings. Generating tailored pairings for each crew also lie in the line of the reviewed literature (see chapter 3). Here, a graph G^k which for each crew k represents the set of pairings tailored for k was widely used. Especially, in the tracking and day-to-day phases for the reasons described above. The pairings given by G^k was the used as columns in a Set Covering Problem or Set Partitioning Problem. Below, a solution approach using these elements will be presented. Firstly, the generation of pairings (G^k) will be covered followed by the selection of the pairings each crew is to man.

5.1 Pairing generation

The goal of the pairing generation is for each crew to generate a set of pairings that she might fly. This is conceptually done by searching through a graph G^k for each crew k which represents the pairings constructed from the open flights which are tailored for k.

Firstly, a more precise definition of how resources are identified will be given, followed by a presentation of a number of heuristics for searching for pairings in G^k.

5.1.1 Resources

A *resource* is a crew that has one or more standby allocated in her schedule. A *resource period* is one or more successive standby periods in a crew schedule. This definition might be extended to blank-days, usable-time and "holes" (described in chapter 2). However, the majority of resource time allocated in the schedule is standby and the extension might not be straight forward due to differences in the rules concerning the use of the different resources. Thus only standby is considered.

On figure 5.1 two resource periods are shown in which crew might be able to cover some open flights. The first period (between t_1 and t_2) is simple; it consists only of 1 standby. An upper bound on the total production time which can be assigned during this period is $t_2 - t_1$. The next period (between t_3 and t_8) is formed by three successive standby. Here $t_8 - t_3$ is an upper bound but not a particularly tight one. A better bound is $(t_4 - t_3) + (t_6 - t_5) + (t_8 - t_7)$, because allthough one may assign production

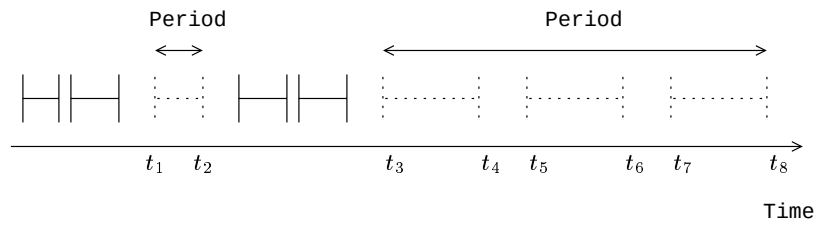


Figure 5.1: *A sample schedule for a crew. Solid lines represents production and dotted lines standby*

between t_3 and t_8 the legality of the production is bounded by the sum o the individual standby.

5.1.2 Pairings

As already described a pairing is a work shift for a crew. The estimate duty time which a pairing takes up is the sum of the length of the dut periods that makes up the pairing (see figure 1.1).

The term *partial pairing* will be used to denote a pairing which is no balanced (Starting and ending and the same base).

5.1.3 Depth first search

The construction of G^k is a way of enumerating all legal pairings for a cre k in a given resource period. As a first attempt a depth first search throug all open flights which lie within the given period (with respect to time) used.

On figure 5.2 a sample search tree is shown. Here the resource is availabl between t_1 and t_2 at base a. Each node in the tree corresponds to a le and each edge to connection time. Each path in the tree, from the root t a leaf, corresponds to a legal pairing to which the crew may be assigned.

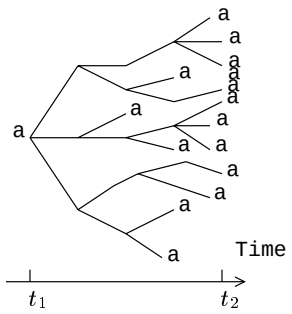


Figure 5.2: *A sample search tree for the depth first search. Nodes corresponds to legs and edges to connection time.*

A closeup on the search tree is shown in figure 5.3. Here four open flights can follow p_i. Two to which it is directly connected, namely, p_j and p_k and two, p_l and p_m, to which it is connected through passive flights k_l and k_m respectively.

On table 5.1 pseudocode for a recursive depth first search which generates pairings is given. As arguments, `DepthFirstSearch` takes the pairing p corresponding to the path to the current node in the search tree, which initially is the empty pairing. Next it takes a list of partial pairings $[l_1, \ldots, l_n]$, which initially consists of one partial pairing for each open flight which matches the given resource period with respect to time. $[l_1, \ldots, l_n]$ is sorted by increasing departure time. Finally, P_r is the set of generated pairings for resource period r.

Line 3-9 check the possible direct connections connecting p with an open flight. This corresponds to the connection between $p_i \mapsto p_j$ and $p_i \mapsto p_k$ on figure 5.3.

Line 10-17 check for possible connections between p and an open flight through a passive connection, which corresponds to $p_i \mapsto k_l \mapsto p_l$ and $p_i \mapsto k_m \mapsto p_m$ on figure 5.3.

Line 19-24 check for passive flights that would balance the current pairing. Which corresponds to node k_n on figure 5.3.

Each of the 3 parts (line intervals) described above have more or less the same structure. Firstly, they check if the given connection should be tried

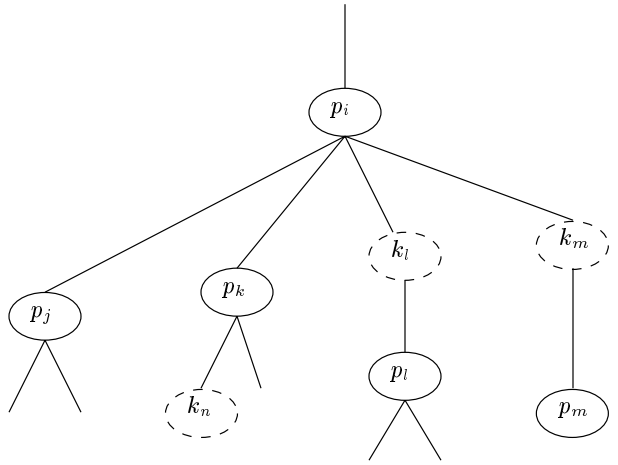


Figure 5.3: *A node p_i in the searchtree with successors p_j, p_k p_l and p_m. p_l and p_m are connected by two passive flights k_l and k_m.*

with the "okto" predicates in line 3, 10 and 19. If the connection shoul be tried they check if a legal pairing has been generated (lines 5, 12 an 21) in which case it is saved (line 6, 13 and 22). If the connection does no result in a legal pairing they check if the subtree beneath the current nod should be explored (lines 7 and 14).

A more in-depth description of the different predicates follows.

`okToAppend(`p`,`l`,`r`)`

`okToAppend(`p`,`l`,`r`)` is a predicate that decides if l should be appended t p when trying to build a pairing for resource r.

On figure 5.4 the situation is sketched. Firstly, it is checked whatever l departure base `c` matches p's arrival base `b` and if the connection tim between p and l, $t_3 - t_2$, lies within a predefined interval. Secondly, it checked whatever l lies within the current resource periods time interva ($t_4 \leq t_5$ and $t_1 \leq t_3$). Finally, it is checked whatever the estimated duty $p \cup l$ is bigger than the estimated duty for the given resource period.

```
1   DepthFirstSearch(p,[l_1,...,l_n],P_r,r)
2   for l_i ∈ [l_1,...,l_n]
3     if okToAppend(p,l_i,r)
5       if isValid(p∪l_i,r)
6         P_r = P_r ∪ [p∪l_i]
7       else if okToContinue(p∪l_i,r)
8         DepthFirstSearch(p∪l_i,[l_{i+1},...,l_n],P_r,r)
9       end
10    else if okToAppendWithPassive(p,l_i,r)
11      k = GetPassive(p,l_i,r)
12      if isValid(p∪[k]∪l_i,r)
13        P_r = P_r ∪ [p∪[k]∪[l_i]]
14      else if okToContinue(p∪[k]∪l_i,r)
15        DepthFirstSearch(p∪[k]∪l_i,[l_1,...,l_n],P_r,r)
16      end
17    end
18  end
19  if okToBalanceWithPassive(p,r)
20    k = GetPassive(p,r)
21    if isValid(p∪[k],r)
22      P_r = P_r ∪ [p∪[k]]
23    end
24  end
```

Table 5.1: *Pseudocode for a depth first search procedure that generates pairings.*

`isValid(`p`,`r`)`

`isValid(`p`,`r`)` checks whatever a pairing p is valid according to union and governmental rules. This is done by a system called RAVE, which SAS currently uses to ensure legality of the schedule. The check is performed by inserting the pairing p into the crew r's schedule and then validate the schedule using RAVE.

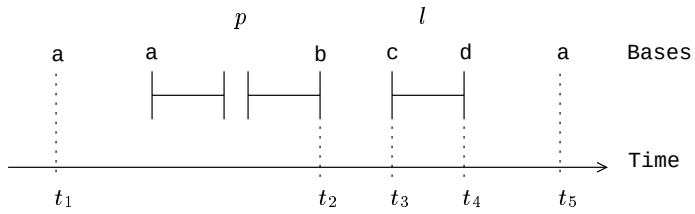


Figure 5.4: *Partially pairing l sought appended to partially pairing p. The corresponding resource period start and end at base* `a` *from time t_1 to t_5.*

`okToContinue(`p`,`r`)`

`okToContinue(`p`,`r`)` checks if the subtree rooted at p should be explore for resource period r. Here a partial legality check is used to ensure tha no illegal node is explored any further. However, since the pairing is onl partial, only a partial legality check can be performed. The pairing p inserted into crew r's schedule, however all activities which lie after p a removed from the schedule before the legality check is performed. This done since the pairing p is not balanced and therefore do not (yet) fit wit the part of the schedule which lies after it.

`okToAppendWithPassive(`p`,`l`,`r`)`

`okToAppendWithPassive(`p`,`l`,`r`)` checks whatever p and l can be connecte through a passive flight. The requirement is that (see figure 5.4) p arriv on a different base than l (`b` $\neq$ `d`), and that the time interval between and l is so "big enough" for a passive flight to be inserted.

`GetPassive(`p`,`l`,`r`)`

`GetPassive(`p`,`l`,`r`)` returns a passive connection to connect p and l if pos sible. This is done by querying the SIG utility for a passive connection.

`okToBalanceWithPassive(`p`,`r`)`

`okToBalanceWithPassive(`p`,`r`)` checks whatever it is realistic to balance pairing p (see figure 5.5) with a passive flight. The requirement is that the time between the arrival at `b` (t_2) and the end of the given resource period (t_3) is big enough for at passive flight and that p is not balanced already (`b` $\neq$ `a`).

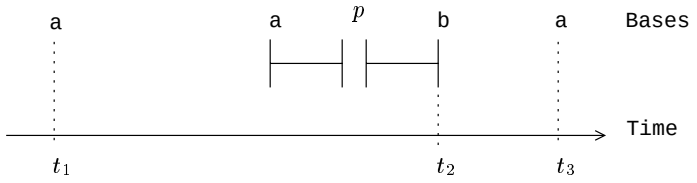


Figure 5.5: *Partially pairing l sought appended to partially pairing p. The corresponding resource period start and end at base `a` from time t_1 to t_5.*

5.1.4 Depth-best first search

In the above description of the depth first search an implicit priority is used during the search because $[l_1, \ldots, l_n]$ is sorted by increasing departure time. This way flights with a small connection times are sought connected before flights with long connection times. This priority might not be optimal. If a cost could be assigned to a pairing then at each node all possible successors could be generated and sorted according to this cost. At each node this would form a searchfront with a given length L_{SF}. The nodes is the searchfront could then be explored in turn by increasing cost.

5.1.5 Pairing cost

Estimating the cost of a pairing is not straightforward. Many factors influence on the quality of a pairing. Currently the only measurement used by SIG is the density of a pairing (introduced in section 4.1.3) which is the total number of flights in the pairing minus the number of passive flights.

This, however, is a very coarse estimate. Therefore a more flexible estimat is introduced.

Together with SIG, four aspects of a pairing have been identified whic influence the quality of a pairing and therefore should be reflected in th cost. These aspects will be introduced below.

Duty

The amount of open flight duty which the pairing covers is important. Th (estimated) duty of an open flight has already been introduced in sectio 5.1.2 as the sum of the duty periods that make up the pairing. To mak this comparable between different resource periods the cost for duty is see relatively to the estimated duty of the period the pairing is intended for:

$$\text{Duty cost} = 1 - \frac{\text{Estimated duty of pairing}}{\text{Estimated duty of resource period}}$$

In other words, this specify what you get (the duty covered by the pairing relative to what you pay (the duty allocated in crew schedule). Since th ratio grows, as the amount of duty that is covered grows the ratio actuall measures what you gain. Therefore 1 minus the ratio is used as the cost. A implicit assumption is made here that the remaining time in the resourc period is not usable. Even though this might not be the case all the tim it seams reasonable during tracking. Here the goal is to use the resourc periods allocated in the schedule and not to build up a schedule for fro scratch.

Passive flights

The passive duty time in a pairing is also important since it basically time where a crew gets payed but do not work. Some passive legs in pairing might be changed to "ordinary" legs, before crew sees the pairing However, this is far from always the case and therefore passive flights shoul be avoided.

Once again a ratio is used to make the measure comparable:

$$\text{Passive cost} = \frac{\text{Estimated passive duty of pairing}}{\text{Estimated duty of resource period}}$$

Problem flight

The notion of problem flight arises because some open flights are more desirable to cover then other, or – put in other words – some open flights are more difficult to cover than other.

A flight from Copenhagen to Aalborg departing at 12.00 is easy to cover compared with a flight from Stockholm to Munich at 23.00. Therefore it is more desirable to cover the second flight than the first at this point in the solution process. This idea is somewhat similar to that of reordering (see section 4.1.2). Here the flights are also ranked according to how big a problem they are. And flights that are bigger problems are sought swapped into the schedule instead of less problematic ones.

Together with SIG the following 4 aspects of an open flight have been identified which characterize a problem flight:

- Time of departure (Early departures are problems).
- Time of arrival (Late arrivals are problems).
- Length of flight (Long flights are problems).
- If flight creates a night stop (Night stops are problems).

Again a ratio is used to make the cost comparable between different resource periods.

$$\text{Problem Cost} = 1 - \frac{\text{Duty of problem legs}}{\text{Estimated duty of resource period}}$$

Crew cover

Crew cover is the average ratio of crew which have resource periods which matches (with respect to time) the legs in the given pairing. This gives a rough estimate of how likely it is that the legs in the pairing can be covered by another crew.

The cost of a pairing is a weighted sum of the introduced costs:

$$\begin{aligned}\text{Cost} =& \text{Duty Weight} \times \text{Duty Cost} + \\ & \text{Problem Weight} \times \text{Problem Cost} + \\ & \text{Passive Weight} \times \text{Passive Cost} + \\ & \text{Crew Cover Weight} \times \text{Crew Cover Cost}\end{aligned}$$

5.1.6 Searching from the "middle and out"

The search strategy forces a given partial pairing to be part of the pai
ings that are generated. This is illustrated on figure 5.6. Here a parti
pairing from `b` to `c` is used as a starting point for 2 depth first searche
One searches backwards in time towards the start of the given period an
another forward in time towards the end of the period.

A modified version of the depth first search described in section 5.1.3
used. The largest modification is that between t_1 and t_2 the search has t
be backwards in time form `b` towards `a`.

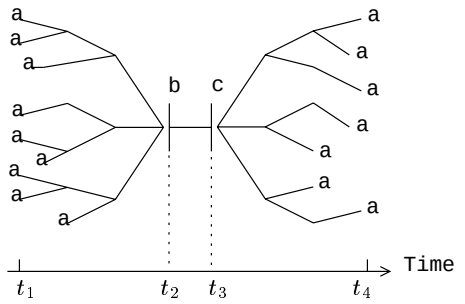


Figure 5.6: *A sample search tree for depth first searching back- and forward from a given pairing.*

5.1.7 Preprocessing of the open flights

When constructing pairings for different resource periods (different crew
several open flights will be sought connected over and over again. Therefo
a preprocessing of the open flights where all possible successors are foun
for each flight might improve performance. This idea forms a clone betwee
the classical approach, where all pairings are generated first and then cre
are assigned, and the idea that is tested in this project where crew a
considered in the generation process.

The idea is to construct two maps which for each leg gives the possibl
direct successors and possible successors through a passive leg. Thought
in terms of graphs the two maps represents a "meta graph" G that contai

all pairings that are going to be considered. The paths represented by the crew specific graph G^k are then a subset of the paths in G. Pseudocode for generating the maps is given in table 5.2.

```
1   ConstructSuccessorMaps([l_1,...,l_n],DS,PS,P)
2   for l_i ∈ [l_1,...,l_n]
3     for l_j ∈ [l_{i+1},...,l_n]
4       if directSuccessor(l_i,l_j)
5           DS[l_i] = DS[l_i] ∪ l_j
6       else if passiveSuccessor(l_i,l_j)
7           PS[l_i] = PS[l_i] ∪ l_j
8           P[(l_i,l_j)] = GetPassive(l_i,l_j)
9       end
10    end
11  end
```

Table 5.2: *Pseudocode for the construction of successor maps for each leg.*

`ConstructSuccessorMaps` takes 4 arguments: The list of open flights $[l_1, \ldots, l_n]$ and 3 maps. DS is a map between a leg l_i and a list of possible direct successors of l_i. PS maps between a leg l_i and the possible successors through a passive leg. P maps a pair of open legs (l_i, l_j) to the corresponding passive leg that can be used to connect l_i and l_j. `directSuccessor(`l_i`,`l_j`)` is a predicate that indicates if l_j is a possible direct successor of l_i. `passiveSuccessor(`l_i`,`l_j`)` is a predicate that indicates if l_i and l_j can be connected through a passive leg.

Now a slightly modified version of the depth-first search from table 5.1 can be used to search the open flights given by the maps.

5.2 Pairing selection

In the previous section a set of pairings has been constructed for each resource period. In this section we focus on the selection of a subset of those pairings which will be the ones actually flown. The problem consists of choosing at most one pairing for each resource period that the corresponding crew has to cover. The problem is modelled as a Set Covering Problem plus some extra constraints. These constraints ensure that a crew is not

assigned two different flights at the same time. This is done by groupin the pairings according to the crew they where generated for (see table 5.3 and then allow at most one pairing to be selected within each group.

The choice of a Set Covering Problem in favour of a Set Partitioning Prob lem rest in the nature of the two problems. Since overcoverage is allowed i the Set Covering Problem, it is always easy to construct a feasible solutio (given that all rows are covered by at least one column each); just includ all columns. On the other hand the Set Partitioning Problem does not a low for overcoverage, and just finding feasible solution might be hard. Th drawback of using the covering model is that some flights might be covere several times.

A mathematical formulation of the problem follows.

$$\text{Maximize} \quad Z = \sum_{i=1}^{m} c_i x_i \tag{5.1}$$

Subject to

$$\sum_{i=1}^{m} a_{ij} x_i \geq 1, \quad \text{for } j = 1, \ldots, n \tag{5.2}$$

$$\sum_{i \in \mathbb{I}_p} x_i \leq 1, \quad \text{for all crew } p \tag{5.3}$$

$$x_i \text{ non negative integer} \tag{5.4}$$

x_i is a binary decision variable that decides whether pairing i is used in th solution or not and c_i is the corresponding cost. a_{ij} indicates if open flight is covered by pairing i. $\mathbb{I}_p$ is the set of indices for columns corresponding t crew p. The first set of constraints ensures that all open flights are covere at least once (5.2). The second set of constraints ensures that each cre contribute with at most one pairing to the solution (5.3).

To ensure that a feasible solution always exists, so called *ghostcrew* a introduced. A ghostcrew has the property that it has exactly one pai ing associated and this pairing covers exactly one open flight. Hence on ghostcrew is introduced for each open flight. Another use of the ghostcre is to control the search process, since the cost placed on a ghostcrew co umn will affect how desirable it is to include non-ghostcrew columns tha covers the same row as the given ghostcrew column.

In table 5.3 an example of a problem instance is given. The pairings for the first crew, crew A, corresponds to the first k_A columns, crew B the next k_B columns and so forth. The last n columns correspond to the ghost crews; one for each open flight forming the union matrix.

c_1	$\cdots$	c_{k_A}	c_{k_A+1}	$\cdots$	$c_{k_A+k_B}$	$\cdots$	c_l	c_{l+1}	$\cdots$	c_{l+n}	
1	$\cdots$	1	0	$\cdots$	0	$\cdots$	1	0	$\cdots$	0	1
0	$\cdots$	0	0	$\cdots$	1	$\cdots$	0	1	$\cdots$	0	1
$\vdots$	$\cdots$	$\vdots$	$\vdots$	$\cdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\vdots$
0	$\cdots$	1	1	$\cdots$	1	$\cdots$	0	0	$\cdots$	1	1

Table 5.3: *An example of an instance of the Set Covering Problem. With n rows corresponding to open flights and m columns to pairings.*

5.2.1 Solving the set covering problem

There are many possible strategies (see [2] for a survey) for solving the Set Covering Problem. One could try to use a linear optimizer[1], one of the many heuristics based on Lagrangian relaxation combined with subgradient optimization or some other heuristics such as metaheuristics.

There are two major problems with the use of a linear optimizer. Firstly, SIG currently has not got one, so if this approach should be taken they would have to buy one. Secondly, the problem which has to be solved will be very large containing between 100 and 1000 open flights (rows) and all from a few thousand up to 150000 pairings (columns). There might be notable savings in time using a heuristic compared with an linear optimizer whit problems of that size. The advantage of getting the optimal solution might be overlooked due to the fact that the cost used only is a rough estimate. The cost is mostly used as a mean of guiding the solution process rather than a precise estimate of solution cost. Therefore the difference between two solutions that have a object value within a few procent might be "invisible" to SAS.

Using Lagrangian relaxation combined with subgradient optimization is affordable measured in running time. But since extra restriction has been

[1]Such as `CPLEX`

introduced into the mathematical model of the standard Set Covering Prob lem, it is not obvious if Lagrangian relaxation could be applied successfull On top of this Lagrangian relaxation is hard to implement. The possib gain would be in solution quality because Lagrangian relaxation is know to outperform other heuristics when solution quality is measured. But a noted above the gain in solution quality might not be worth the trouble.

The desired characteristics point in the direction of a (meta)heuristi Firstly, they are able to solve large problems with a reasonable solutio quality very fast. Secondly, they are often easy to implement. It has bee possible to find a set of successful applications of Simulated Annealing o the set covering problem and therefore Simulate Annealing is chosen.

In table 5.4 pseudocode for the simulated annealing [10, 1] heuristic is given As arguments `SimulatedAnnealing` takes the Set Covering Problem **SCP** the start temperature T_0, the cooling factor α and maximum number o iterations i_{max}.

```
1   SimulatedAnnealing(SCP, T_0, α, maxIter)
1   X_B = ∅
2   cost(X_B) = ∞
4   X = generateSolution(X_B, SCP)
3   for i ∈ 0, ..., i_max and not stop()
6     X' = searchNeighbourhood(X, SCP)
7     δ = cost(X') − cost(X)
8     ϖ = random[0, 1[
9     if δ < 0 or ϖ < e^(−δ/T_i)
10      X = X'
11      if cost(X') < cost(X_B)
12        X_B = X'
13      end
14    end
15    T_(i+1) = α × T_i
16  end
```

Table 5.4: *Simulated annealing.*

Solution and neighbourhood

A solution $\mathbf{X}$ is a selection of columns (pairings). $\mathbf{X}$ is feasible if all flights are covered at least once and if there exist no group in which to columns are selected.

The neighbourhood of a solution $\mathbf{X}$ is defined as the set of solutions that can be obtained by randomly dropping d columns from $\mathbf{X}$ and rebuilding the solution without reusing the dropped columns. The constraint that the dropped columns can not be reused only applies to non-ghostcrew columns. This ensures that the neighbourhood does not contain infeasible solutions. If it was possible to forbid the selection of a ghostcrew column after it had been dropped, it would not be guaranteed that a feasible solution could be constructed. The problem arises if a ghostcrew column is dropped where the matching row can not be covered by any other column.

Solution generation

`generateSolution(X,SCP)` uses a greedy heuristic to generate a solution given a, possible empty, solution $\mathbf{X}$ and the problem **SCP**. The function is sketched on table 5.5. It selects at random an uncovered row j (line 3) and calculates for each column i that covers j the ratio between the number of uncovered rows covered by i and the cost of i (line 7-9). The column i that maximizes this ratio is included in the solution (line 12). This process continues until a feasible solution is obtained.

Note that only those columns i that does not fulfill the predicate `okToSelect(`i`,X,SCP)` are considered in lines 6-9. There are two cases where it is not allowed to include column i:

- If another column which is in the same group as i already is selected (which corresponds to constraint (5.3) from the mathematical formulation) or
- if column i is a non-ghostcrew column that has just been dropped from the solution (which corresponds to the constraint introduced in the definition of the neighbourhood).


```
1   generateSolution(X,SCP)
2   while not feasible(X)
3      j = selectUncoveredRow(X,SCP)
4      ratio = −∞
5      i_max = −1
6      for i ∈ isCoveredBy(i,j) and okToSelect(i,X,SCP)
7         if uniqueCovers(i) / cost(i) > ratio
8            ratio = uniqueCovers(i) / cost(i)
9            i_max = i
10        end
11     end
12     X = X ∪ i_max
13  end
```

Table 5.5: *A greedy approach for solution generation.*

Stopping criteria

The function `stop()` (in table 5.4) represents the stopping criteria. Th simplest possible may be a fix number of iterations. Even though th criteria might seam poor it is often used in conjunction with some othe criterion. Choosing a maximum number of iterations is often used to pr determine an upper limit on the solution time.

If the maximum number of iterations is chosen large enough it is almos always the case that the solution process can be stopped before this numbe of iterations is reached. A natural measure to apply is to look at the las L iterations and observe how the object value has varied. If no or litt variation has been observed the solution process is halted.

Chapter 6

Implementation

This chapter gives a short overview of the implementation. The program has been implemented as an extension to TAP-AI. The advantages using this approach are many; firstly, SIG's existing data structures and library functions can be used. This way pairings, flights, schedules, crew etc. are already implemented and ready to use together with several library functions where the most important are the legality checking functions. Secondly, this makes it easy to extract data from SAS and use it to test the program. The drawback is that TAP-AI has been developed within several different programming paradigms: It started out as a PROLOG program, then some of the code was ported to C and lately an object oriented approach has been used using C++. Nowadays, the program can run completely without using the parts of the program that still resides in PROLOG.

As a natural choice an object oriented approach has been taken given that SIG currently has used this paradigm in the development of TAP-AI.

There has been written around 9000 lines of code in this project. The code and test data are available at IMM at `/home/proj/proj7/public`.

In the program code for TAP-AI the term *sling* is used which is a synonym for *pairing*. Hence, to keep the terminology consistent sling has been used instead of pairing in the code. However, to avoid confusion in this report the term pairing is used on the subsequent figures.

6.1 Program design

On figure 6.1 the relations between the main pairing generation objects ar given. Two main components are the representation of resources and th generation of pairings. For each crew a resource class is given (`CrewResour` which contains a linked list of the resource periods where the crew ma work (`CrewResourcePeriod`). The pairing generation heuristics are im plemented through a Strategy pattern [5]. Therefore an abstract pai ing generator class has been made (`PairingGenerator`). This way ne pairing generators can easily be added: They just have to implemen the `PairingGenerator` interface. Hence, implementing and experimen ing with the different heuristics described in section 5.1 has been mad easy. On figure 6.1 the inheritance from the `PairingGenerator`-class to th `PairingGeneratorDepthFirst`-class (implementing the depth-first searc from section 5.1.3) has been shown. Similar inheritances have been mad for classes implementing the other generation heuristics.

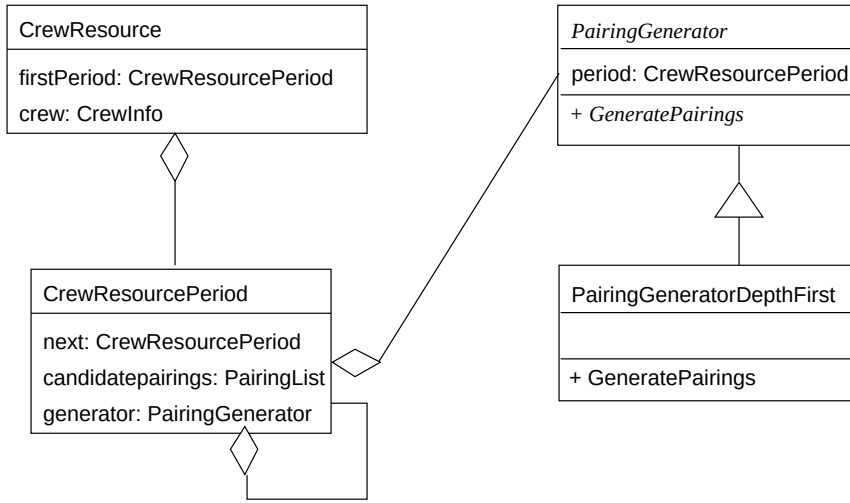


Figure 6.1: *UML-diagram for the classes representing the resources and the associated pairing generation heuristic.*

On figure 6.2 a diagram is given showing the relation between the tw

classes that represent the Set Covering problem and the class that represents the Simulated Annealing heuristics that is used the solve the Set Covering Problem. The problem is split into two classes: One that represents the problem (`SetCoveringProblem`) and one that represents a solution (`SetCoveringSolution`). Hence all data common to different solutions to the same problem are kept in the problem class.

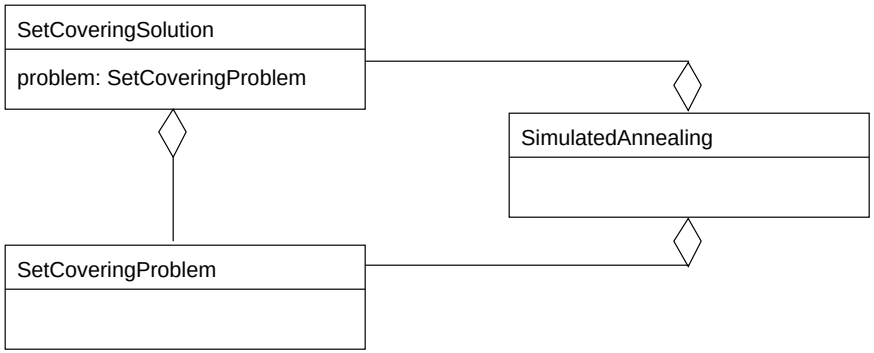


Figure 6.2: *UML-diagram for the classes representing the Set Covering Problem and the Simulated Annealing heuristic.*

6.2 Memory consumption

During the implementation a high memory usage has been observed (in some situations above 1 GB), which points in the direction of a memory leak.

For the part of the code which has been written in this project some leaks where found *and* corrected. Some testing has been carried out to detect further leaks in the added code, however, no further sign of leaks could be found. One leak was found in the original code from SIG; unfortunately SIG has not yet had the time to close the leak.

The high memory usage still persist and as will be described in the following chapter several test have been aborted due to high memory consumption.

Chapter 7

Results

In this chapter the software will be tested on "real life" data from SA
Firstly, a description of the different data sets used will be given. This
followed by a description of the tuning of the parameters which are pa
of solution process, such as simulated annealing parameters, weights in th
cost function and so forth. Finally, the different heuristics are tested an
compared.

The tests have been carried out at IMM on `serv2` and `serv3` which bot
are HP servers model `J7000` running HP-UX 11i. They each have 4 44
Mhz CPU's and 4 GB memory.

7.1 Data

The data sets used in this chapter consists of "real world" snapshots take
from SAS's schedule. A snapshot of the schedule gives the present status
the schedule at the point in time it is taken. The current set of open flight
is given and for each crew the current activities assigned are given. Befo
the actual data is presented a short description of three important facto
will be given.

Crew category describes the type of personal to consider. There are tw
meta categories *cabin* and *flight deck*. Within each meta categor

there are several different sub categories. As described earlier this project focuses on cabin crew and therefore the three main types of cabin crew are included in the data. This is reasonable because often a crew is qualified to work as all or many of the different sub categories within one of the meta categories.

Time interval is the period of time the snapshot covers. If a small interval is chosen the problem can be solved "fast" because the number of open flights and the number of crew are proportional with the interval length. But a small interval might not be solved "good" because the continuity of the problem. Many open flights might lie close to or cross the endings of the interval which makes them impossible to cover.

Time before operation is the number of days between the day the snapshot was taken and the first day in the snapshot.

In all 16 different snapshots have been taken. The shots have been divided into 4 groups according to the time interval they cover.

On table 7.1 four snapshots are listed each covering a time interval of 2 days. The first column shows the name of the shot, the second column the time before operation, the third the dates (in February) that are covered by the shot. The two last columns shows the number of crews and the number of open flights respectively.

Name	Time before operation	Date	Crew	Open flights
1	1	6.-7.	226	168
2	2	7.-8.	226	198
3	7	12.-13.	297	133
4	14	18.-19.	334	215

Table 7.1: *Data snapshot from 5th February 2002. Each shot covers 2 days.*

On figure 7.1 the distribution of resource periods on different time intervals is shown for the four snapshots from table 7.1. The figure shows how the data close to operation (snapshot 1 and 2) has many short resource periods and few long. The data sets with 7 and 14 days before operation (snapshot 3 and 4) has fewer short resource periods and several long periods. This

situation arises because as operation gets closer, resources in schedule ar used to close open flights reducing the number of resource periods and th length of the remaining resource periods in schedule. The "top" in the "20 25"-interval is caused by the fact that SAS often allocates resources tha have this length.

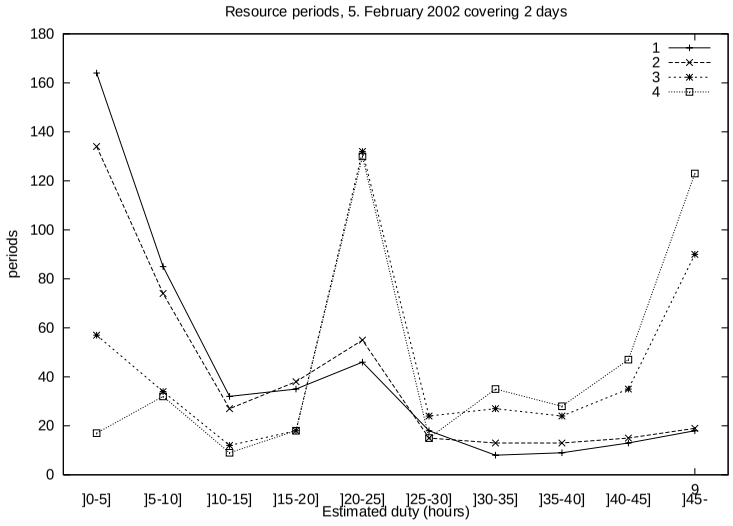


Figure 7.1: *Number of resource periods versus resource periods length for the data set from table 7.1.*

Similar to the presentation of snapshots 1, 2, 3 and 4 snapshot 5, 6, 7, and 9 are listed in table 7.2 and the distribution of resources periods shown on figure 7.2.

Name	Time before operation	Date	Crew	Open flights
5	2	7.-9.	294	245
6	5	10.-12.	331	212
7	8	13.-15.	358	285
8	14	19.-21.	456	209

Table 7.2: *Data snapshot from 5th February 2002. Each shot covers 3 days.*

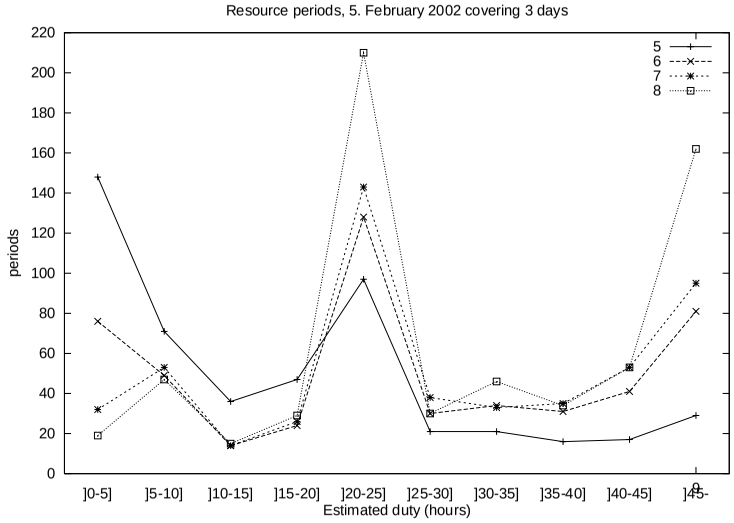


Figure 7.2: *Number of resource periods versus resource periods length for the data set from table 7.2.*

Snapshots that covers 4 and 5 days are listed below in table 7.3 and table 7.4 with corresponding resource periods distributions plot on figure 7.3 and figure 7.4 respectively.

Name	Time before operation	Date	Crew	Open flights
9	2	18.-21.	469	354
10	6	22.-25.	510	716
11	10	26.-1.	483	642
12	14	2.-5.	520	447

Table 7.3: *Data snapshot from 15th February 2002. Each shot covers 4 days.*

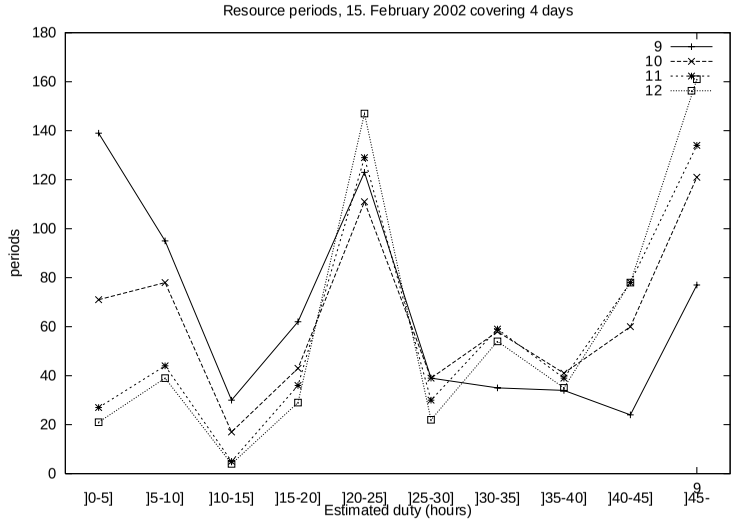


Figure 7.3: *Number of resource periods versus resource periods length for the data set from table 7.3.*

Name	Time before operation	Date	Crew	Open flights
13	2	18.-22.	553	502
14	6	22.-26.	563	899
15	10	26.-2.	555	751
16	14	2.-6.	611	512

Table 7.4: *Data snapshot from 15. February 2002. Each shot covers 5 days.*

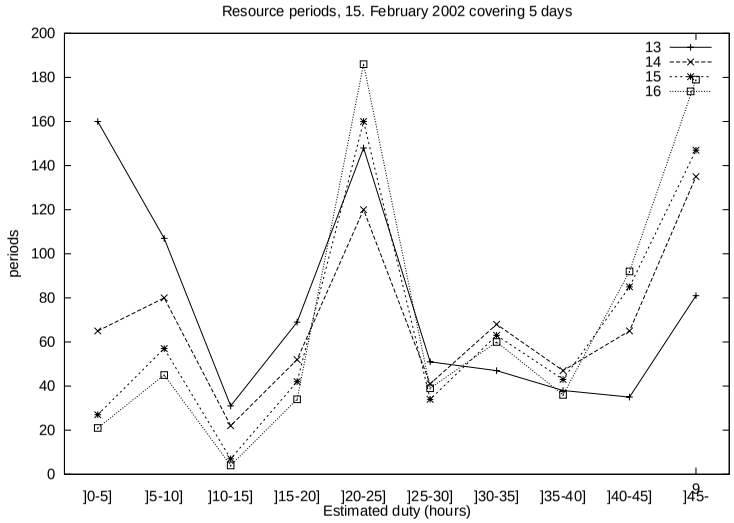


Figure 7.4: *Number of resource periods versus resource periods length for the data set from table 7.4.*

7.2 Model parameters

The effect on the solution from the choice of values for the different parameters which is part of the model are tested and appropriate values are found.

For several reasons these tests has only been carried out on a limited se of the small snapshots. Firstly, the long running times ranging from couple of hours for the small shots and more than 10 hours for some of th big shots makes testing extremely time consuming. Secondly, due to th memory consumption (as described in section 6.2) it has not been possibl to run tests with some combinations of snapshot and parameters. This – c course – introduces further uncertainty regarding the choice of parameter for the large snapshots. Hopefully, some similarity should exist among th snapshots and the parameters found when testing on the small snapshot should provide good indications for the choices of parameters for the larg snapshots.

7.2.1 Parameters in Simulated Annealing

When using Simulated Annealing several parameters (such as start tem perature, cooling factor and plateau length) can be adjusted and the per formance of the heuristic depends heavily on how well these parameters ar tuned. In the literature it is often reported that "the best parameters wa found after many experiments". This is due to the fact that the parameter are heavily problem dependent.

Start temperature and cooling factor

As hinted in the pseudocode for the Simulated Annealing procedure th cooling scheme is not a "plateau schedule", where the temperature is con stant in L successive iterations and lowered every L iterations. The reaso is that if values for L, T_0 and α are determined, a similar behaviour ca be achieved by setting $L = 1$ and raise α. Therefore the cooling schec ule sketched in the pseudocode, where the temperature is lowered at eac iteration, is used and only T_0 and α has to be estimated.

On figure 7.5 an average solution value is shown versus start temperatur for three different cooling factors. For each choice of cooling factor an start temperature the average is taken over the 4 data sets from table 7. (using depth first search for pairing generation).

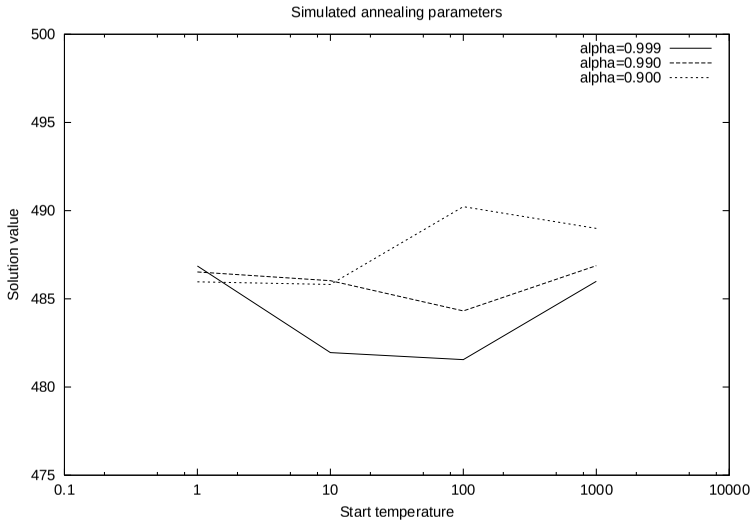


Figure 7.5: *Solution value versus start temperature T_0 for three different cooling factors α.*

Figure 7.5 shows that $\alpha = 0.9$ and $\alpha = 0.99$ cools to fast and best results are achieved with $\alpha = 0.999$. The choice of temperature is not that clear; 1 is to low and 1000 is to high but 10 and 100 both seem to be acceptable.

To show the impact from T_0 on the solution process two runs are shown with different choices of T_0. On figure 7.6 the object values for the current solution and the best found are shown versus the number of iterations. As one can see many worse solutions are accepted within the first 3500 iterations which indicates that the solution process moves around in the solution space. Above 5500 iterations the search converge towards a local search where only better solutions are accepted.

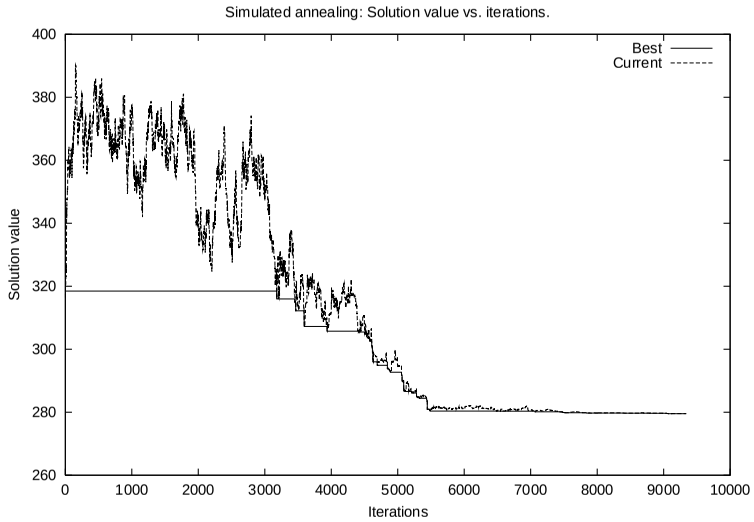


Figure 7.6: *Simulated annealing. $T_0 = 100$ and $\alpha = 0.999$.*

On figure 7.7 the start temperature is lowered. Now worse solutions are onl accepted on a *very* small scale after the first 700 iterations. And above 300 iterations the solution process looks almost like pure local search. Thus th solution space is not explored sufficiently and the solution found is wors than the one found on figure 7.6 with the higher temperature.

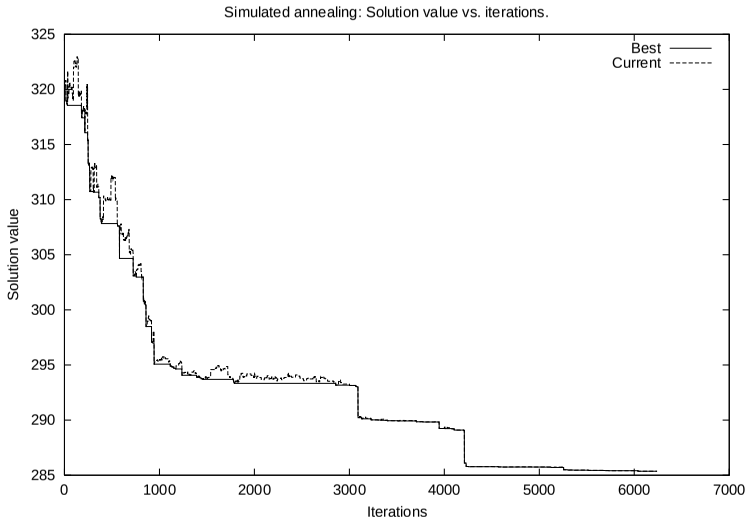


Figure 7.7: *Simulated annealing.* $T_0 = 1$ *and* $\alpha = 0.999$.

Another indicator, often considered when visualizing simulated annealing, is the acceptance rate. The acceptance rate is the ratio between the total number of accepted neighbouring solutions and the total number of neighbouring solution considered (for some iteration interval). The acceptance rate corresponding to figure 7.6 and figure 7.7 is shown on figure 7.8 and figure 7.9 respectively.

A rule-of-thumb states that the accept rate should be in the interval 30%-70% in the "first number of iterations". Considered that this only is a rule-of-thumb the solution process depicted on figure 7.8 seems fine, even though an initial acceptance rate of 80% is observed. In contrast the acceptance rate of the solution process depicted figure 7.9 decreases very fast and indicates further that a initial temperature of $T_0 = 1$ is too low.

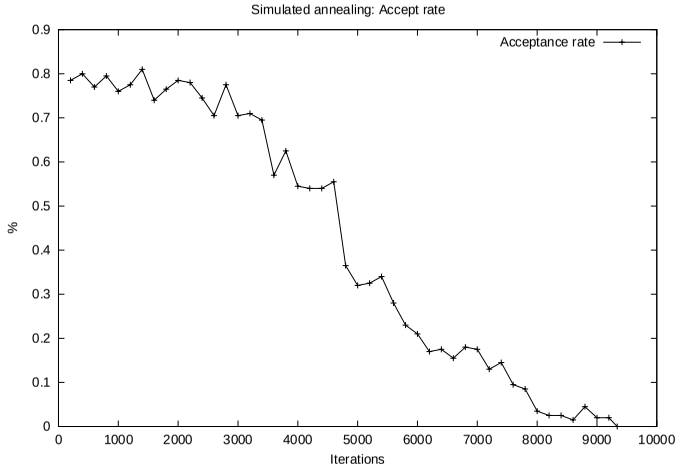


Figure 7.8: *Simulated annealing.* $T_0 = 100$ *and* $\alpha = 0.999$.

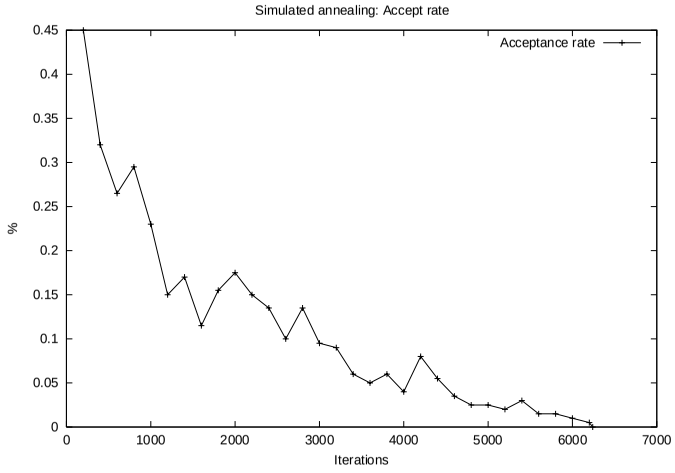


Figure 7.9: *Simulated annealing.* $T_0 = 1$ *and* $\alpha = 0.999$.

Neighbourhood

The important parameter when dealing with the neighbourhood, is the number of columns d which is dropped from the current solution, when searching the neighbourhood. On figure 7.10 the plot shows the influence on the solution value from different choice of d. As one can see, a choice between 3 and 6 seems reasonable. Above 6 the plot shows that the disruption of dropping more than 6 columns, when searching the neighbourhood, results in a fluctuated solution value.

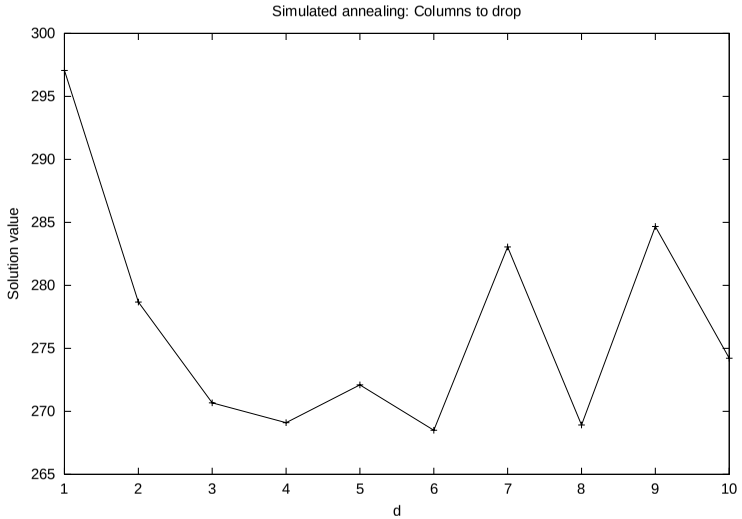


Figure 7.10: *Solution value versus d*

Unfortunately, the first limited testing showed that a value of $d = 2$ was sufficient. Thus a larger number of experiments was carried out using $d = 2$. However, as the above graph shows a value of 3 or 4 should have been chosen. To be able to use the results from the first large set of experiments, all subsequent experiments was also run using $d = 2$.

Modification of the accept function

During experiments with simulated annealing an unfortunate property o the combination of the neighbourhood and the accept function (line 9 o table 5.4) has been observed. If $\delta = 0$ the neighbouring solution will alway be accepted since $e^{-\frac{0}{T_i}}$ is 1 for all positive values of T_i. To overcom this, the accept function is modified so the neighbouring solution only accepted if the "old" accept function is satisfied *and* if the object value o the neighbouring solution is different from the object value of the curren solution. The reason that $\delta = 0$ is that many columns has the same pairin cost, because many are the same.

7.2.2 Ghostcrew cost

There are two reasons for the use of ghostcrew: Firstly, it ensures that feasible solution to the Set Covering Problem can be found, and secondl – yet another – mean of controlling the tradeoff between coverage of prob lem and non-problem flights. When the cost of a ghostcrew is altered affects how attractive it becomes to include the corresponding column i the solution. A high ghostcrew cost makes it attractive to search for a nor ghostcrew column that covers the row covered by the ghostcrew column. low ghostcrew cost decreases the gain of excluding the ghostcrew colum from the solution. By assigning different costs to two ghostcrew columns can be made more profitable to cover one of the corresponding rows wit a non-ghostcrew column than the other.

Below there will be distinguished between two types of ghostcrew columns Problem ghostcrew columns covering rows corresponding to problem flight and ghostcrew columns covering rows corresponding to non-problem flights

On the following three figures (figure 7.12, 7.11 and 7.13) an average ove snapshot 1, 6 and 13 have been used.

On figure 7.11 the ratio between the number of problem flights covered an the total number of open flights covered is plotted versus the differenc between ghostcrew cost and problem ghostcrew cost for five different choice of ghostcrew cost. As an example the solid line represents a choice of for ghostcrew column cost. When the ghostcrew cost difference equals the cost of a ghostcrew column equals a problem ghostcrew column an approximately 23% of the open flights covered are problem flights. Sti

using the solid line as a reference point, raising the difference between the cost ghostcrew columns and problem ghostcrew columns to 3 makes the ratio of problem flights raise to 56%. Since the cost of non-ghostcrew columns are kept constant it becomes increasingly profitable to use non-ghostcrew columns that covers problem flights instead of the corresponding problem ghostcrew columns.

As can be seen it is possible to affect the ratio by increasing the difference between ghostcrew cost and problem ghostcrew cost when the ghostcrew cost is low (1 and 2). However, if the ghostcrew cost becomes large (especially 8 and 16) the cost difference between ghostcrew columns and problem ghostcrew columns has no effect. At this point the cost of both ghostcrew columns and problem ghostcrew columns have become so high that it always is desirable to cover the rows by including non-ghostcrew columns.

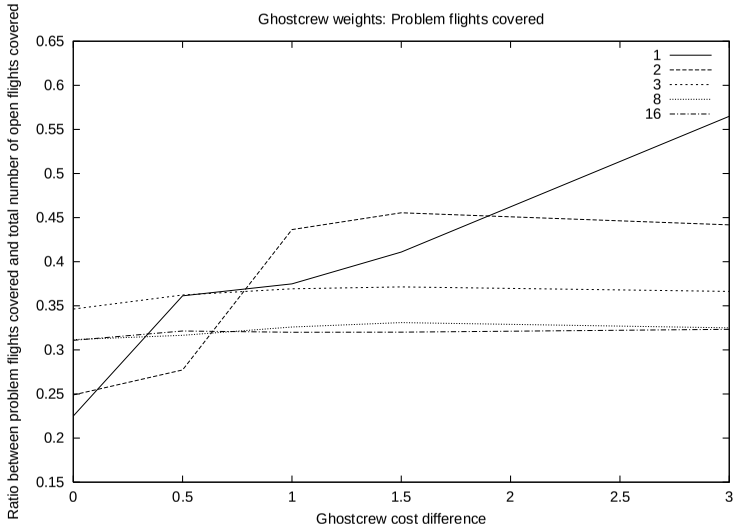


Figure 7.11: *Ratio between problem flights covered and open flights covered versus difference between ghostcrew cost and problem ghostcrew cost.*

Another effect that the cost of ghostcrew column has is on the ratio between open flights covered and total number of open flights. This is illustrated on

figure 7.12. Again using the solid line as a reference point and a ghostcre cost difference of 0, 40% of the open flights are covered. As the difference increased, still considering the solid line, an increase in open flights covere is observed.

Each horizontal line represents a choice of ghostcrew cost; as the ghostcre cost increases, more and more open flights are covered. This lies in line what one would expect: As the cost on ghostcrew increases it gets mo profitable to use non-ghostcrew columns to cover the corresponding ope flights (rows). When the cost of ghostcrew columns becomes high, all no ghostcrew columns are included and further increase of ghostcrew colum cost does not result in a higher open flight coverage. This is illustrated b the two top most horizontal lines representing ghostcrew cost of 8 and 1

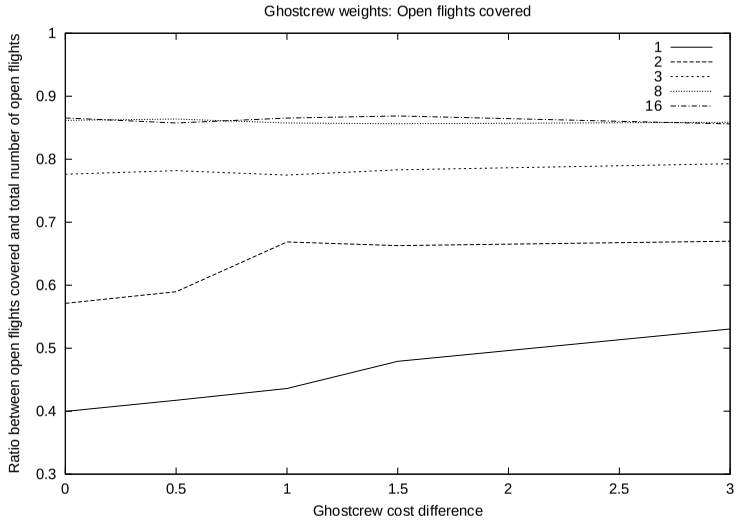


Figure 7.12: *Ratio between open flights covered and total number of open flights versus difference between ghostcrew cost and problem ghostcrew cost.*

The price payed when increasing the ghostcrew costs is an increased use passive flights. This is illustrated on figure 7.13; if the ghostcrew cost small (1 and 2) an initial low use of passive is obtained. As the differen

between ghostcrew columns and problem ghostcrew columns is increased the use of passive increases as well. Again, when the ghostcrew cost becomes high (8 and 16) the cost of ghostcrew columns is so high that it is always profitable to cover the rows by including non-ghostcrew columns. Hence the ratio of passive used becomes constant.

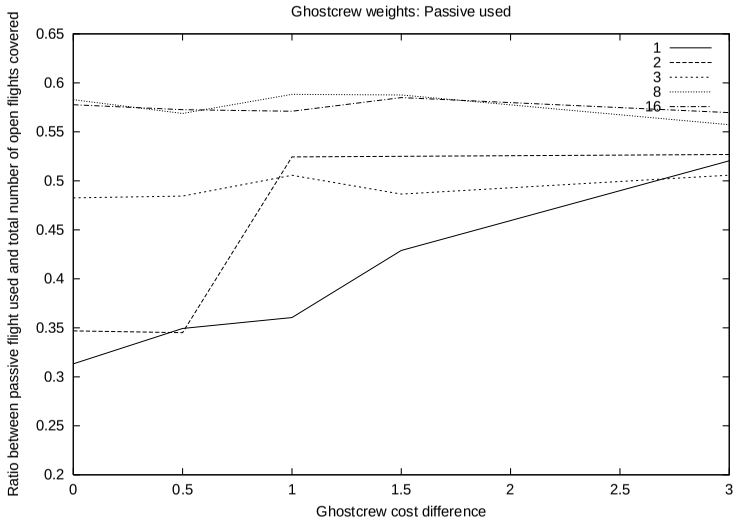


Figure 7.13: *Ratio between passive flights used and open flights covered versus difference between ghostcrew cost and problem ghostcrew cost.*

7.2.3 Pairing cost weights

Another mean of controlling the solution process is the weights used in the cost function (described in section 5.1.5). The two most correlated weights are the problem weight and the passive weight; if one wants to cover many problem flights one has to pay a price which usually increases the ratio of passive flights used. Firstly, the problem and passive weights will be examined followed by an examination of the utilization weight and the crew cover weight.

As test data snapshots 1 and 5 are used and the numbers reported are a average over the two.

Problem and passive weights

On figure 7.14 the ratio between problem flights covered and total numbe of open flights covered is plotted versus problem weight for five differer choices of passive weight. As expected the ratio of problem flights covere increases as the problem flight weight increases.

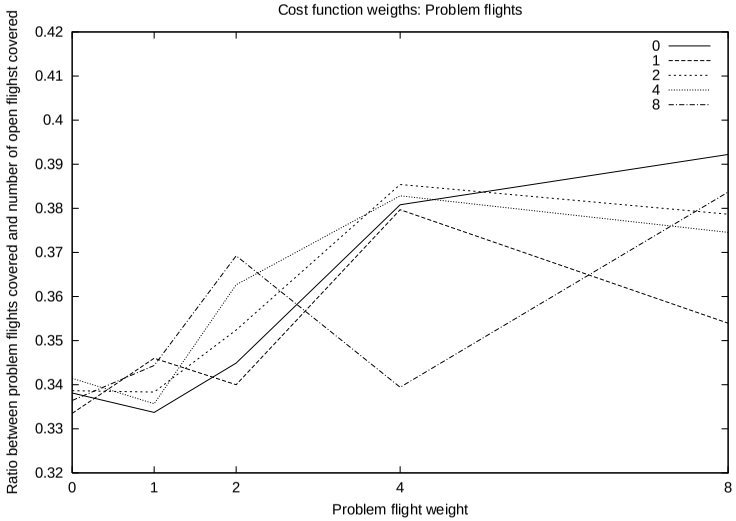


Figure 7.14: *Ratio between problem flights covered and open flights covered versus problem flights weight. For five different passive weights.*

On figure 7.15 the use of passive flights is plotted versus the problem weigh for five choices of passive weight. Firstly, it is noticed that a small ove all decrease in the ratio of passive used can be observed as the proble flight weight increases. One might expect that the use of passive flight would increase as more and more problem flights are covered. However,

decrease in the number of open flights covered as the problem flight weight increases (as shown on figure 7.16) dominates the use of passive flights. Therefore, a slight decrease is observed on figure 7.15. It is also noted that the five horizontal lines are (roughly speaking) nicely distributed according the their passive weight. The topmost, solid line represents the lowest passive weight of 0 and results in the highest use of passive. As the passive weight is increased for each line, the corresponding use of passive drops.

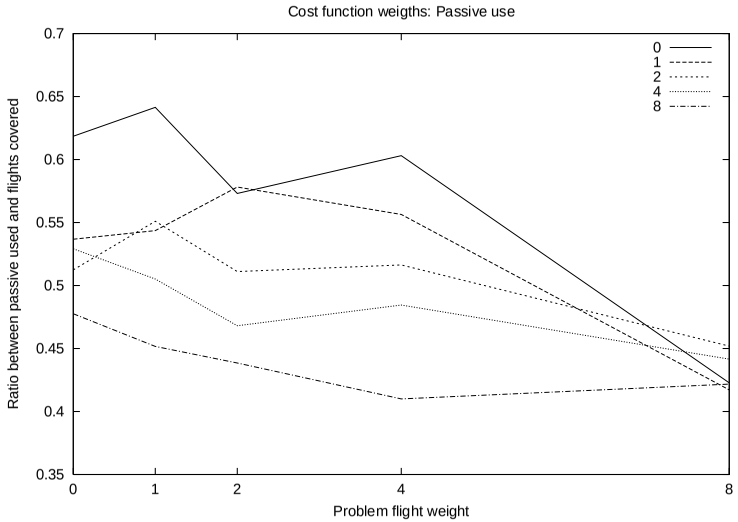


Figure 7.15: *Ratio between passive flights used and flights covered versus problem flights weight. For five different passive weights.*

On figure 7.16 the ratio between open flights covered and the total number of open flights is plotted versus problem flights weight. The plot shows how the ratio of covered flights decrease as the problem flight weight increases. Since columns that covers no or just a few problem flights are punished harder and harder as the problem flights weight increases, it becomes more and more desirable to cover the corresponding rows by either using other non-ghostcrew columns that covers many problem flights, or to use ghostcrew. As the ghostcrew costs are kept constant, a huge decrease

in open flight coverage is observed for large values of the problem flight weight. At this point only the non-ghostcrew columns that cover a large number of problem flights have a cost that is low enough to be include into the solution.

The current section describes one of the two possible ways the solution process can be guided, namely, by adjusting the weights in the object functio (cost function). The previous chapter showed the other way by adjustin the cost of ghostcrew columns. An interesting comparison can be mode between figure 7.16 and figure 7.12 both showing how the ratio of open flight covered is affected by adjusting either problem flights weight or ghostcre cost. With an initial low problem flight weight (on figure 7.16) around 90 of the open flights are covered. At this point the cost of non-ghostcre columns is so low that it is always desirable to include them into the solu tion. Another way of obtaining approximately the same effect is to mak the cost of ghostcrew columns high, which is illustrated on figure 7.1 with a high ghostcrew column cost a open flight coverage of around 87% achieved.

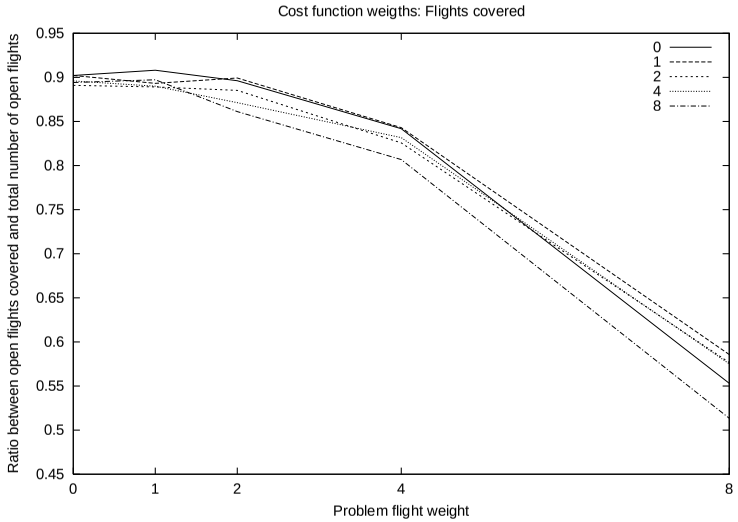


Figure 7.16: *Ratio between open flights covered and total number of open flights versus problem flights weight. For five different passive weights.*

Utilization weight

The utilization weight is used to control "how well" pairings fit into the resource periods they are intended for. On figure 7.17 the influence from the utilization weight on the open flight coverage is shown. As expected the coverage decreases as the utilization weight increases, since the cost is increased for non-ghostcrew columns which "fits" purely into the resource periods they are intended for. Hence it becomes more profitable to use non-ghostcrew columns that have a good utilization of their corresponding resource periods or to use a ghostcrew column.

Accordingly, the use of passive decreases which is shown on figure 7.18.

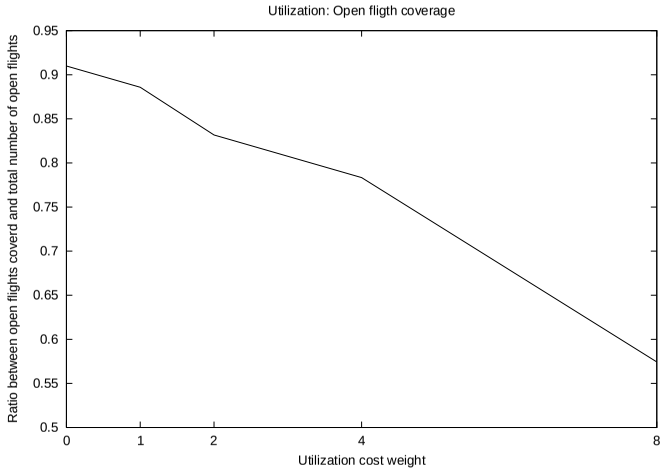


Figure 7.17: *Ratio between open flights covered and total number of open flights versus the utilization weight.*

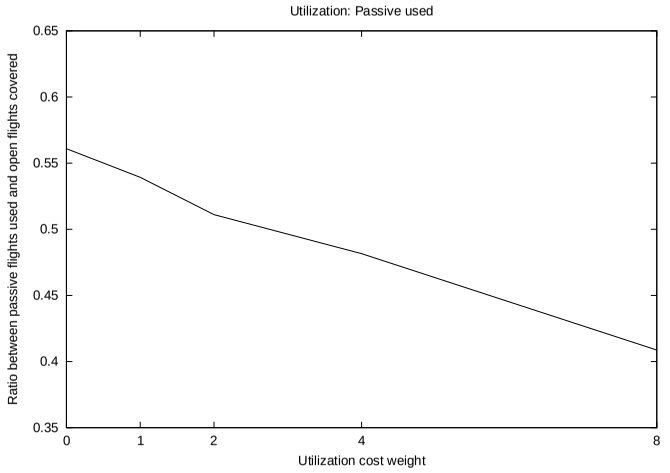


Figure 7.18: *Ratio between passive flights used and number of open flights covered versus the utilization weight.*

Crew cover weight

The crew cover weight punishes those pairings which cover open flights with a high crew cover (as described in section 5.1.5). As the crew cover weight is increased the open flights coverage decreases as it gets more and more profitable to use ghostcrew columns. This is shown on figure 7.19.

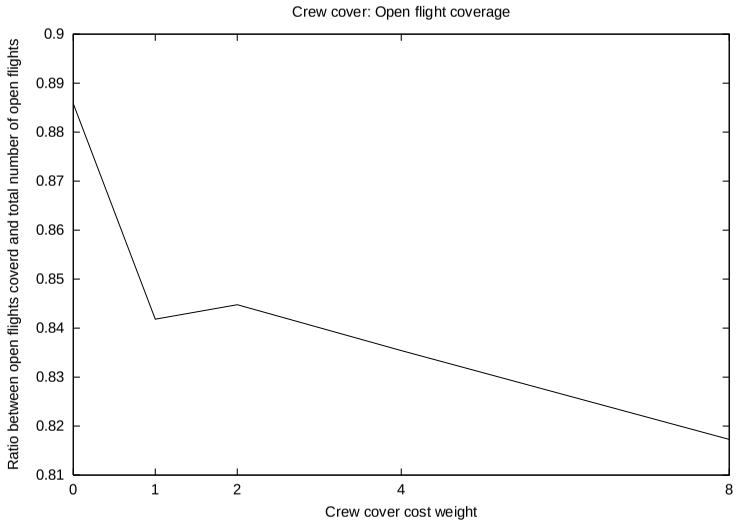


Figure 7.19: *Ratio between open flights covered and total number of open flights versus the crew cover weight.*

On figure 7.20 the corresponding use of passive is shown. The use increases because the flights that becomes more profitable to cover are the ones with a low crew cover, which often are flights which are hard to cover (often problem flights). This is illustrated on figure 7.21 where the ratio of problem flights covered are plotted. Here a small increase is observed.

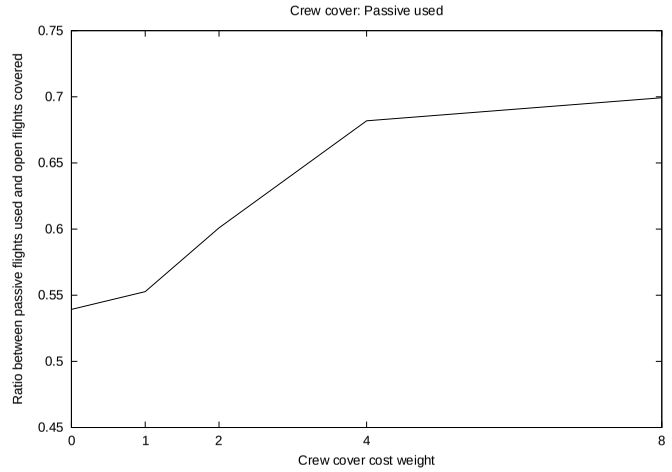


Figure 7.20: *Ratio between passive flights used and number of open flights covered versus the crew cover weight.*

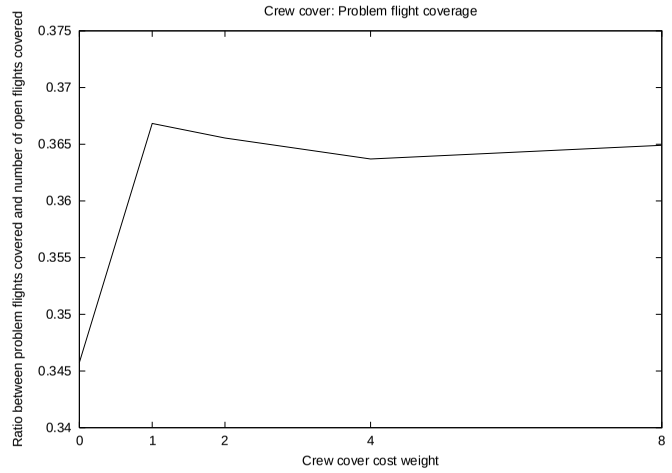


Figure 7.21: *Ratio between problem flights covered and number of open flights covered versus the crew cover weight.*

7.2.4 Bounding the depth first search

During the first few pilot experiments it became clear that the search tree had to be bounded more aggressively than described in the pseudocode (see table 5.1). Here the bounding is made by ensuring time and destination match and by use of the legality checking utility. However, this does not provide an efficient limitation of the search tree. Below the new bounds are introduced and tested.

After a few pilot experiments it turned out that for some search trees several hundred of thousand nodes where explored, which becomes *very* time consuming. A simple solution is an upper bound on the number of search nodes to explore, below called M_1. Another idea, which also is based on observations from the pilot experiments, is an upper bound on the number of nodes to explore without any generation of legal pairings, below called M_2. For some crew a huge number of nodes are explored without any successful pairing generation. The reason being that a "full" legality check cannot be carried out before a complete pairing has been constructed. Since the nodes in the search tree (besides the leaf-nodes) are all partial they are not legal because they are not balanced. Therefore only a limited legality check can be used and the violation of the rule set for some pairings will first be discovered when they become balanced.

The test has been carried out on snapshots 1 and 5 and the results are the average over these two snapshots.

On figure 7.22 the influence on the solution time from the choices of M_1 and M_2 are shown. The decisive factor for the solution time is clearly M_1; the choice of M_2 has no significant influence on the overall running time.

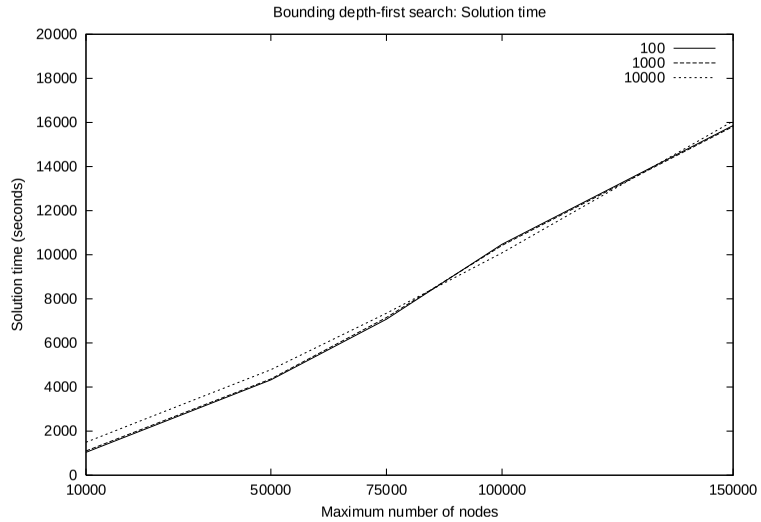


Figure 7.22: *Solution time versus M_1 for three different choices of M_2.*

To get a corresponding measure of the influence on the solution quality th solution cost is plotted on figure 7.23 and the ratio of open flights covere in the final solution is plotted on figure 7.24.

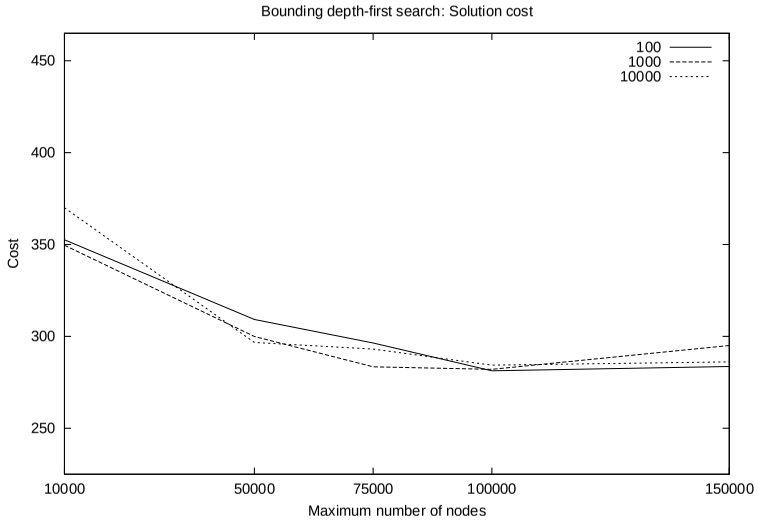


Figure 7.23: *Solution cost versus M_1 for 3 different choices of M_2.*

Again, the most significant factor seem to be M_1. However, choosing a high value of $M_2 = 10000$ allow for a smaller choice of $M_1 = 75000$. This might be interpreted as follows; by increasing M_1 you clearly generate more pairings allowing for a better solution to the Set Covering problem. However, lowering the value of M_1 might be compensated by increasing the value of M_2. Now, instead of generating many pairings for some crew, several other crew which was earlier cut off by M_2 now has pairings generated.

Another measure of the solution quality is the ration of open flights covered, which is plotted on figure 7.24. The plot corresponds well with figure 7.23; around $M_1 = 75000$ the plot seem to flatten out especially for the two high choices of M_2.

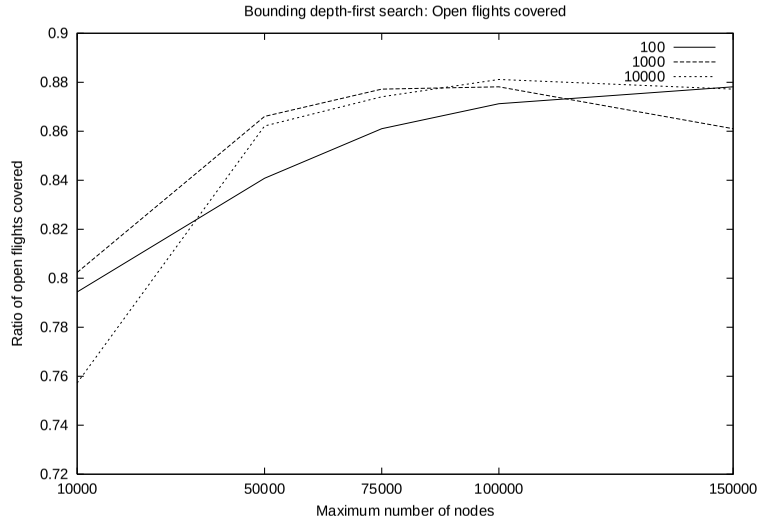


Figure 7.24: *Ratio of open flights covered in final solution versus M_1 for three different choices of M_2.*

7.2.5 Searchfront length for depth-best first search

The important parameter when using depth-best first search is the lengt L_{SF} of the searchfront. The search is limited to the L_{SF}-best at each nod A small value of L_{SF} will result in a very greedy search that only explore the currently best partial pairings thereby lowering the solution qualit notable. On the other hand a large value of L_{SF} will cause the search t explore close to all successors of a given node making the search a plai (unbounded) depth first search and loosing the desired performance gain

On figure 7.25 the influence from L_{SF} on the running time is shown. A one would expect the running time increases with L_{SF}; more and mor nodes are explored.

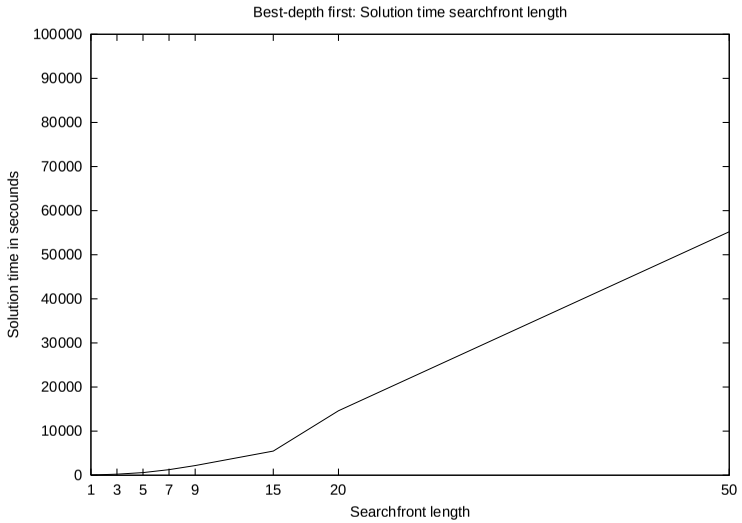


Figure 7.25: *Depth-best first search; solution time in seconds versus L_{SF}.*

Figure 7.26 shows how the solution value is affected by L_{SF}. As expected a huge gain in solution quality is obtained as L_{SF} increases up to 20. For $L_{SF} > 20$ the solution quality is not improved much when L_{SF} is increased.

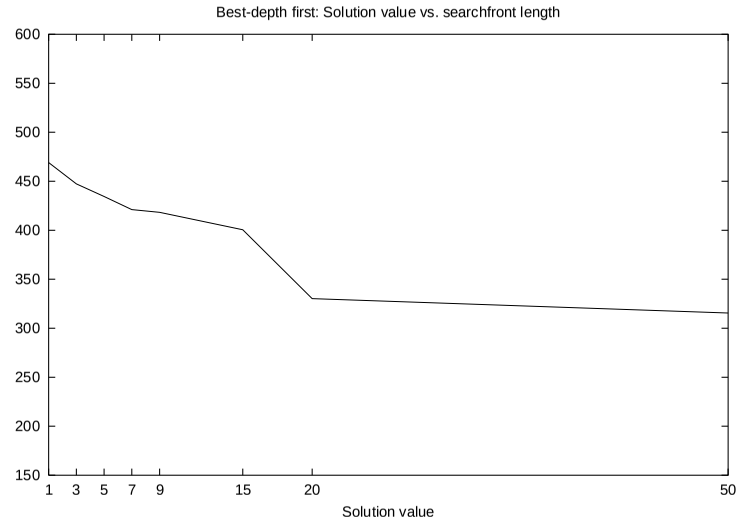


Figure 7.26: *Depth-best first search; solution value versus L_{SF}.*

Another important measure for the solution quality is the number of ope flights which are covered in the final solution. Therefore the number o open flights covered versus L_{SF} is plotted on figure 7.27. Again the plo behaves as expected; for $L_{SF} > 20$ the graph levels out.

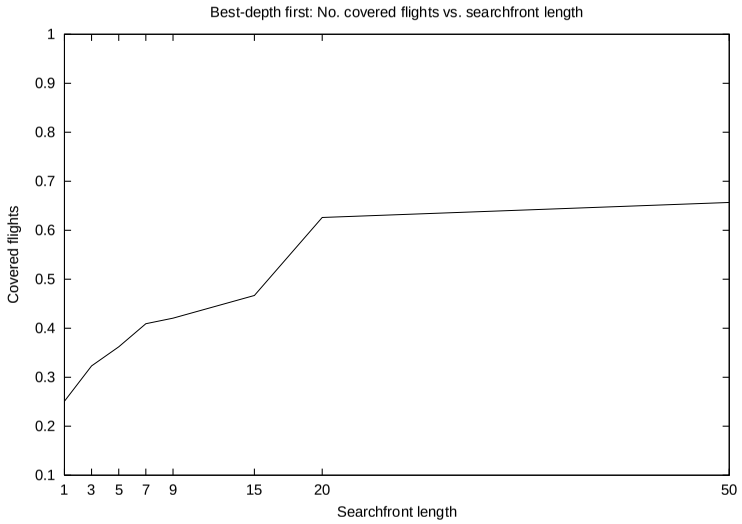


Figure 7.27: *depth-best first search. Number of covered open flights in final solution versus L_{SF}.*

As indicated above a value of $L_{SF} = 20$ shows to be appropriate. Here time savings are achieved without affecting the solution quality notable.

7.3 Performance tests

In this section a comparison of the different heuristics developed will be carried out, along with a comparison these against SIG's paring generation mode 1 (as described in section 4.1.3).

Using SIG's paring generator instead of one of the developed heuristics is easy: Firstly, SIG's generator is used to generate pairings from the open flights. Secondly, for each crew the subset of the generated pairings that she might man are identified. This establishes a situation similar to the one obtained from the other generation heuristics. Hence, the Set Covering Problem can be initialized and solved as if one of the other heuristics had been used.

Below abbreviations for the different pairing generation heuristics will b used. These are given in table 7.5.

Pairing generation heuristic	Abbreviation
Depth first search	DFS
Depth-best first search	DBFS
SIG's pairing generation mode 1	SIG1
Depth first search with preprocessing	PRE

Table 7.5: *Heuristics and corresponding abbreviation.*

7.3.1 Searching from "middle and out"

The last heuristic described in chapter 5 was the searching from "the mi dle and out" as it was described in section 5.1.6. However, this heurist will not be tested any further in this chapter, due to the fact, that the cu rent setup is not sufficient. The heuristics tries to build a pairing given partial pairing, but in the current setup no appropriate way of choosing constructing this partial pairing is available. A small number of pilot test have been carried out, where the problem flights have been used as startin points. However, this approach produces the same pairings over and ov again and some kind of pre-pairing has to be introduced to produce som appropriate starting points for this heuristic.

7.3.2 DFS versus DBFS

In this section DFS and DBFS will be compared. For the larger snapshots length of $L_{SF} = 20$ is not sufficient to bound the search tree: It becom too time consuming and for some of the snapshots, the memory limit 1 GB is exceeded. Therefore, the two bounds introduced for the DFS a also applied to the DBFS. The parameters used are $L_{SF} = 20$, $M_1 = 5000$ and $M_1 = 10000$. However for the DFS a choice of $M_1 = 25000$ has bee necessary for most of the large snapshots (10, 11, 12, 14, 15 and 16) simpl to stay below the 1 GB memory limit.

On figure 7.28 the open flight coverage is compared for DFS and DBF The expected gain in solution quality do not seem to appear; only a sma

improvement is seen for the large shots, however, DFS has been run with a lower value of M_1. Hence the observed gain is even less impressive. For the small shots DBFS performs worse than DFS.

A 100% flight coverage is not expected since a fraction of the open flights lie close to either the start or the end of the snapshot. Such flights may be impossible to cover given the chosen time interval.

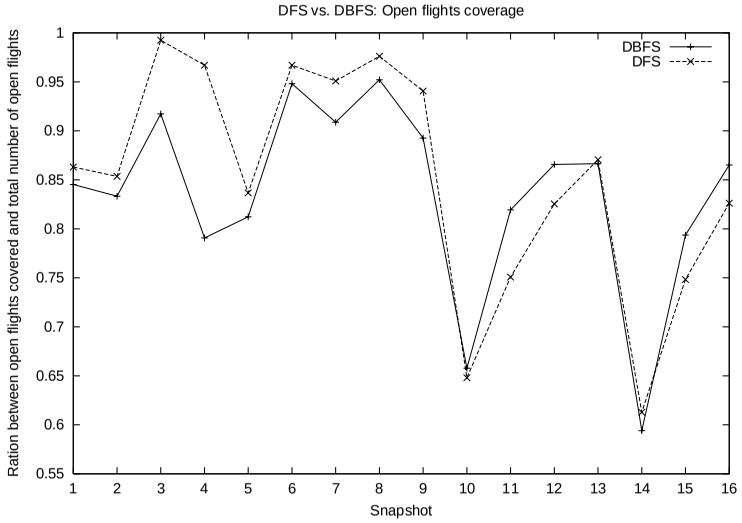


Figure 7.28: DFS *versus* DBFS*; the plot shows the ratio between open flights covered and total number of open flights for the 16 snapshots.*

On figure 7.29 the solution time for the two heuristics is shown. For the large snapshots a comparison is hard to carry out since the lower value of M_1 has been used for DFS. Hence, the big difference in solution time is seen for snapshots 10 and above. For the small shots it seems like the overhead introduced by sorting the searchfront is larger than the gain obtained by bounding the searchfront.

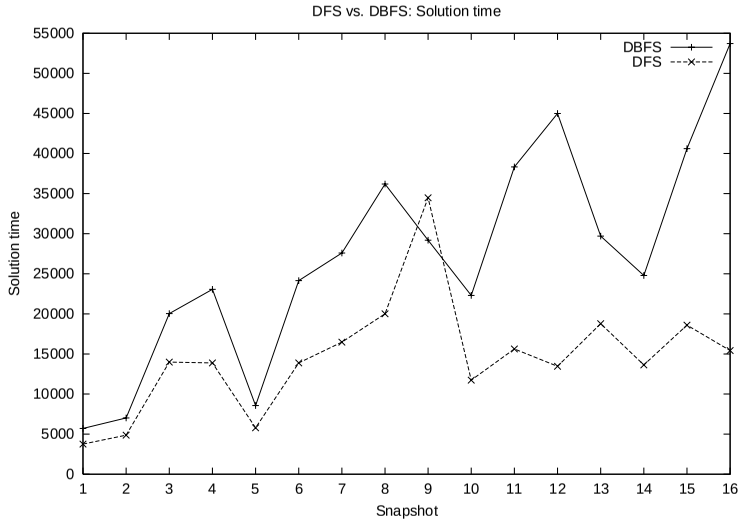


Figure 7.29: DFS *versus* DBFS*; solution time versus snapshot number. 40000 seconds* $\sim$ *11 hours.*

The use of passive flights is shown on figure 7.30. Roughly speaking lower use of passive flights can be observed for the DBFS than the DF This seems reasonable since the use of passive is represented in the co function. Hence, the DBFS explore nodes, which represents a low use passive before nodes representing a high use of passive.

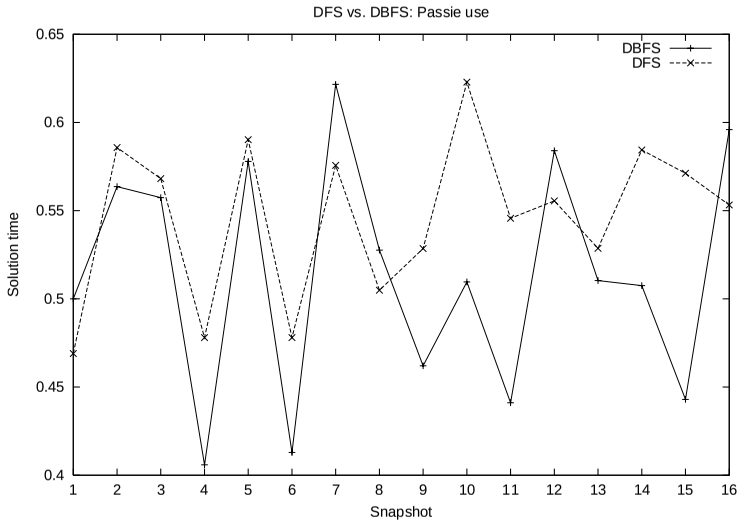


Figure 7.30: DFS *versus* DBFS*; ratio between passive use and number open flights covered versus snapshot number.*

The above tests shows, that the DBFS does not give the quality improvement in the solution to be expected. The main reason for this being that the cost function has been constructed with a full pairing in mind. An example could be: A partial pairing which (at the start of the search tree) consist of a passive flight only would be punished hard by the cost function. It might not even be explored, even though this partial pairing might turn out to be part of a "good" full pairing.

7.3.3 DBFS versus SIG1

On figure 7.31 the open flight coverage is compared for the DBFS and SIG1 heuristics. As it can be seen, the DBFS results in a notably higher coverage compared with the SIG1. Only for one snapshot, number 14, the SIG1 results in a higher open flight coverage. As shown in table 7.4 snapshot 14 contains the highest number of open flights which may explain the poor performance. This behaviour is also observed for snapshot 10 which contains the third

highest number of open flights. The reason being that the DBFS is bounde if the number of open flights are large, larger search trees have to be buil to include more flights. However, this is permitted by the bounds.

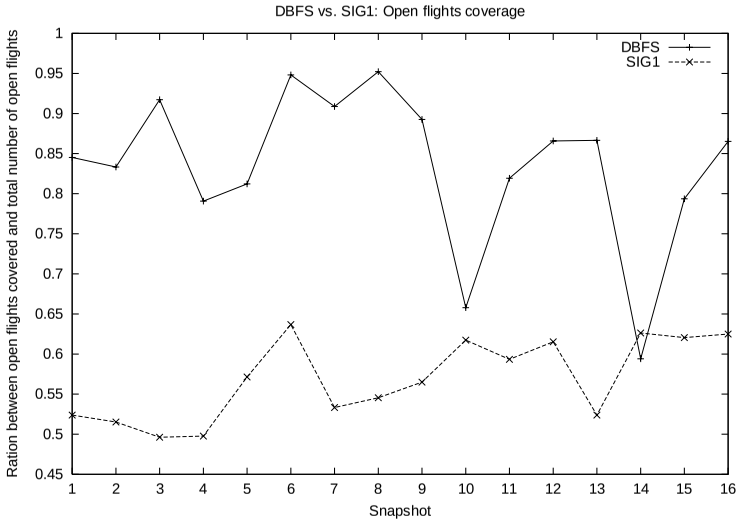


Figure 7.31: DBFS *versus* SIG1 *pairing generation mode 1; the plot shows the ratio between open flights covered and total number of open flights for the 16 snapshots.*

A small increase in the coverage for SIG1 can also be observed as the snap shot number increases. As the snapshot number increases, the size of th snapshot measured in time interval also increase. As shown in section 7. the number of large resource intervals increases as the time interval i creases. So SIG1 performance better on snapshots where the resources a less fragmented rather than ones, where resources are more fragmente This is also consistent with the description given in chapter 4: The prob lem with SIG1 was, that the pairings generated was to long to fit the sma fragmented resources.

Both snapshot 9 and 13, which are taken just 2 days before operatio results in high coverage for the DBFS. At this point, resources are ver fragmented and the tailoring of pairings seems to payoff.

Two main prices are payed for the gained coverage: Firstly, the running of SIG1 can be carried out in a few minutes while the DBFS runs for several hours. This is shown on figure 7.32. For the two snapshot giving a low coverage, the solution time is corresponding low. This is caused by a low solution time for the Set Covering Problem: Since the shots contains many open flights, the introduced bounds results in fewer generations of legal pairings, which again result in a smaller Set Covering Problem, which can be solved faster but with a worse result.

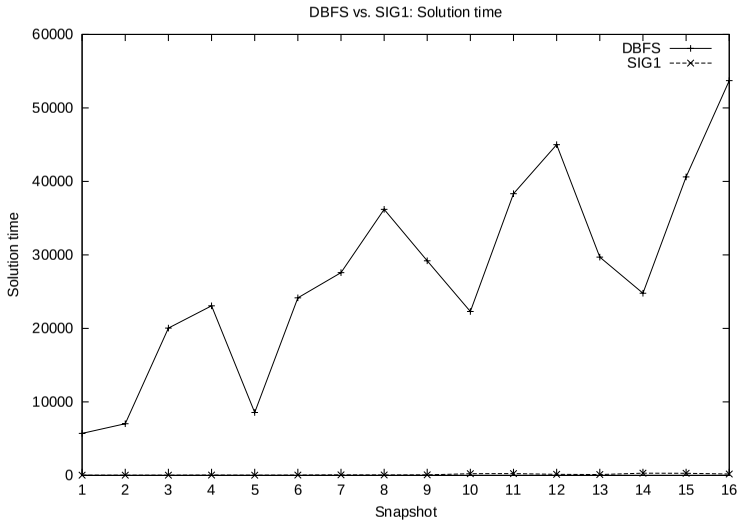


Figure 7.32: DBFS *versus* SIG1 *pairing generation mode 1; solution time versus snapshot number. 40000 seconds* $\sim$ *11 hours. Due to the big difference in running time* SIG1 *only appears as a straight line just above the x-axes.*

Secondly, the use of passive is also increased which is shown on figure 7.33. This might be caused by DBFS covering more problem flights, which would require a higher use of passive. Thus, on figure 7.34 the problem flight coverage is shown for the two heuristics. As it can be seen, a notable higher problem coverage is observed for the DBFS.

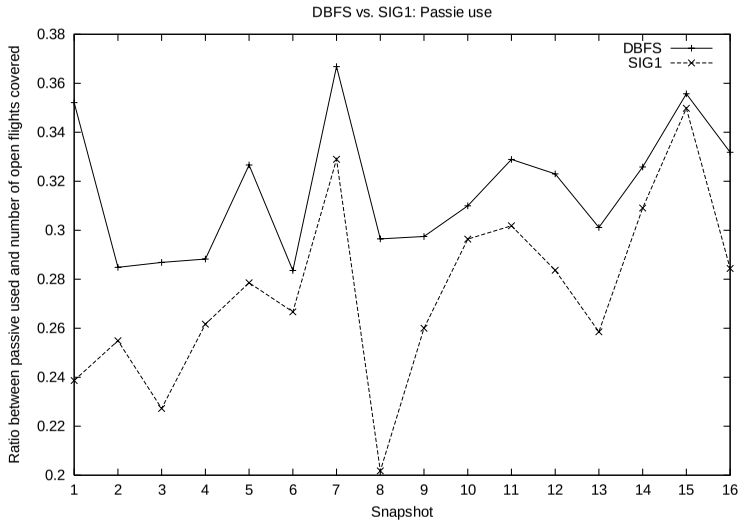


Figure 7.33: DBFS *versus* SIG1 *pairing generation mode 1; ratio between passive use and number of open flights covered versus snapshot number.*

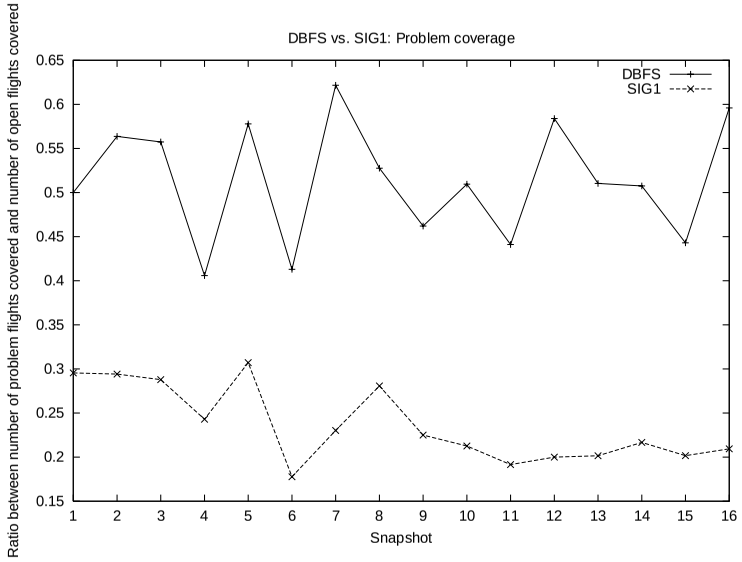


Figure 7.34: DBFS *versus* SIG1 *pairing generation mode 1; ratio between problem flights covered and number open flights covered versus snapshot number.*

7.4 Preprocessing

As seen above, one of the major limitations by using the heuristics is the running time. As a first attempt this has been targeted by introducing a preprocessing of the open flights as described in section 5.1.7. The results are described below.

On figure 7.35 the open flights coverage is shown for the four described heuristics. As can be seen the introducing of preprocessing increases the open flights coverage for the large snapshots. The preprocessing reduces the search space by excluding all non-legal connections from the search. Thus, more nodes can be investigated and more legal pairings can be generated resulting in better solutions to the Set Covering Problem.

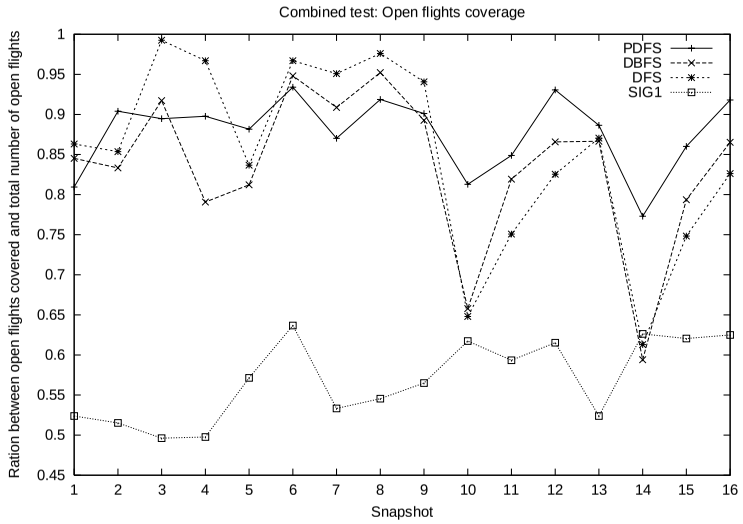


Figure 7.35: *Open flights coverage versus snapshot number for the four heuristics.*

The solution time is shown on figure 7.36. In this figure a notable improve ment can be seen; PRE terminates before 20000 seconds (= 5.5 hours) fo all snapshots except shot 10, which is just above 20000 seconds. Compare with a solution time for some of the large snapshots of above 11 hours fo the DBFS this result must be considered satisfying. Note that M_1 has bee further lowered for DFS on the large snapshots as described earlier. Henc a relatively small solution time is observed for snapshots numbers large than 10.

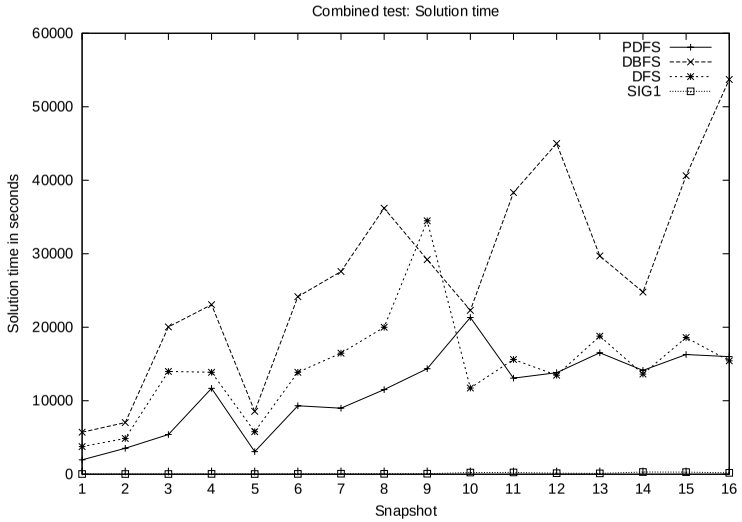


Figure 7.36: *Solution time versus snapshot number for the four heuristics.*

The ratio between the time used for generating pairings and the total solution time is shown on figure 7.37. As it can be seen DFS uses above 90% of the time generating pairings and thus less than 10% of the time solving the Set Covering Problem. PRE – on the other hand – uses around 65% of the time generating pairings and thus 45% of the time is used for solving the Set Covering Problem. As it was shown on figure 7.36 DFS and PRE had approximately the same solution time for the large snapshots, hence, the numbers are comparable. DFS uses a large period of time generating pairings, however the resulting Set Covering Problem is solved fast with poor results (as earlier shown). This is mainly caused by the size of the Set Covering Problem which for the PRE becomes around 150000 columns for some of the large snapshots, while DFS only results in Set Covering Problems of sizes around 30000 columns.

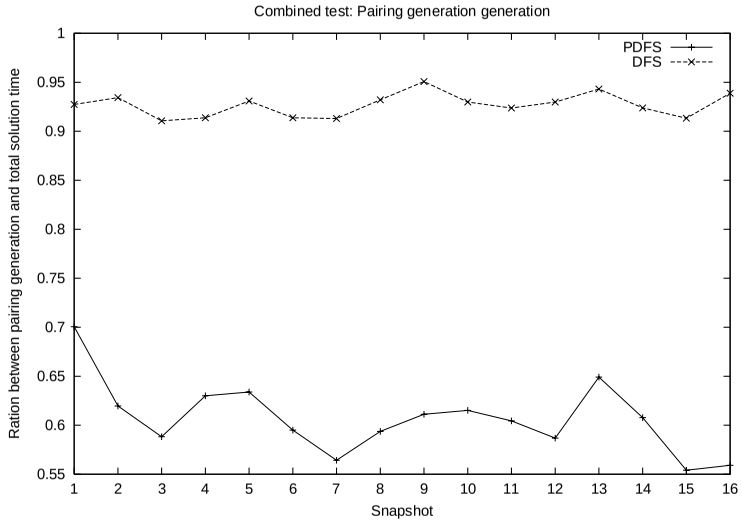


Figure 7.37: *Ratio between time used solving the Set Covering Problem and time used generation pairings versus snapshot number for* PRE *and* DFS.

Looking at the use of passive, a small improvement can be seen on figu 7.38, which shows the ratio between number of passive used and numb of covered flights. Again the improvement must be ascribed to the mo exhaustive exploration of the search tree.

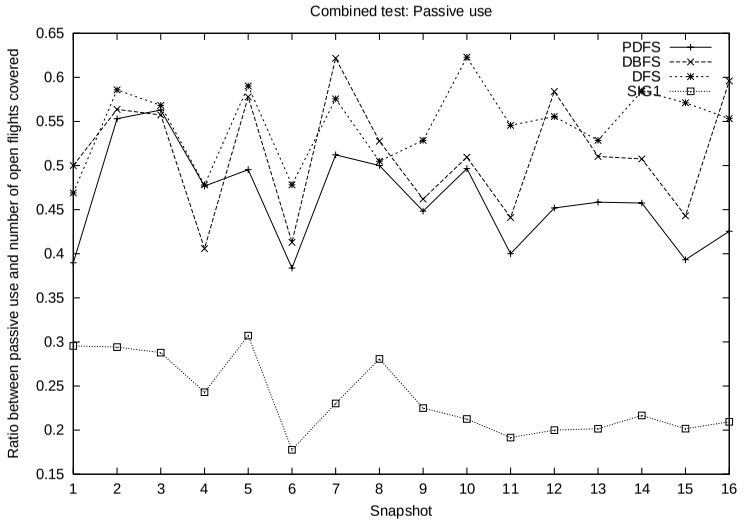


Figure 7.38: *Ratio between passive and number of open flights covered versus snapshot number for the four heuristics.*

To illustrate the corresponding impact on the problem flight coverage, the problem flight coverage versus snapshot number is plotted on figure 7.39. Here it is seen, that the problem flights coverage is not decrease for the PRE, even though a small decrease in passive use was observed on figure 7.38.

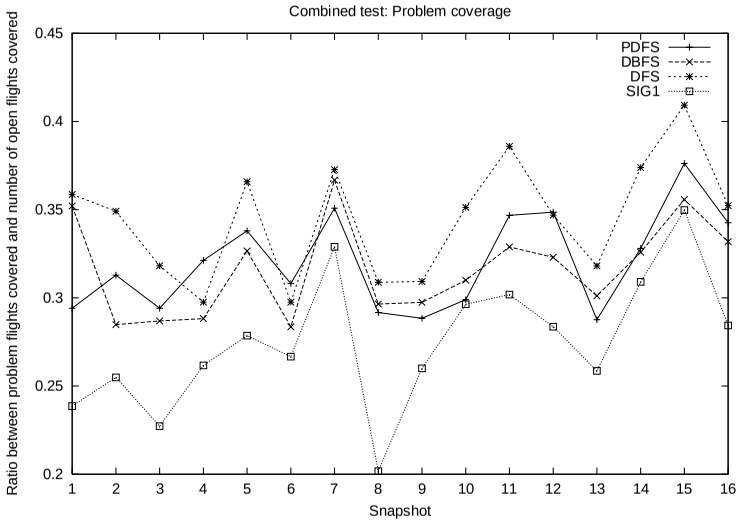


Figure 7.39: *Ratio between problem flights covered and number of open flights covered versus snapshot number for the four heuristics.*

Chapter 8

Conclusion

The main goal of this project has been to construct, implement and test a solution approach for the airline crew scheduling problem during the tracking phase, with the starting point of the software currently used by SAS. In addition, a "crew oriented" approach should be tested, where knowledge about the availability of crew, which, at some point in the future might man a pairing, should be used during the generation of the pairing.

A review of the literature showed, that the (published) models dealing with the tracking and day-to-day phases mostly uses a "crew oriented" pairing generation scheme combined with the solution of a Set Covering Problem or Set Partitioning Problem. However, all the models described were formulated and solved using some kind of linear optimizer. Due to the problem size and the current setup given by SAS, such formulation and solution approach was not a first choice.

Therefore, a solution approach using the "crew oriented" idea and the solution of a Set Covering Problem was developed within the given framework of the existing software at SAS. Three different pairing generation heuristics were successfully implemented together with a heuristic for solving the Set Covering Problem. A fourth generation heuristic was implemented but never tested due to reasons described in section 7.3.1. Finally, a measure for the quality of a pairing was developed.

The numerical results showed a substantial increase in the number of open flights covered using the new heuristics compared with the software cur-

rently used by SAS. Two main prices are payed for this improvemen Firstly, a large increase in the use of passive flights is observed. Howeve the increase in passive use has only been compared with the results of SAS heuristic, which covered a far smaller number of open flights – naturally yielding a lower use of passive. During the project an early personal eva uation from SAS stated that the use of passive flights"seemed high", an unfortunately, any further evaluation has not been possible so far.

Secondly, a large increase in the running time was observed: Where the SA heuristic running time can be measured in minutes the running times o the implemented heuristics is measured in hours. However, with a plannin horizon of a couple of days, a night of calculations might not be too high price to pay for a notably improved solution. A first attempt was made i relation to reducing the running time. This was done by introducing som preprocessing to the pairing generation, eliminating infeasible connection once and for all. Hereby, a notable improvement in solution time and small improvement in solution quality were achieved.

In addition, a pairing quality measure has been introduced and tested wit satisfactory results: It is possible to guide the solution process so tha certain properties (use of passive, problem flight coverage, open flights cov erage etc.) can be weighted to produce different types of solutions.

8.1 Outlook

With the results of this work at hand, a thorough evaluation of the result especially the quality of the generated pairing should be made in collabo ration with SAS.

In addition, several other areas might be targeted during further worl Firstly, a reduction of the running time. One approach, which has alread been tried with success, was to apply some preprocessing. This was don by making some generic choices once and for all instead of repeating ther over and over again for each crew. The preprocessing applied uses legalit However, more restrictive choices based on the arrival and departing base of flights might also be used.

Secondly, an improvement of the solution quality should be targeted. I the given framework, feedback from the set covering module might be use in the pairing generation module. Instead of generating a large number o

pairings at once, a limited number of pairing could be generated and the resulting Set Covering Problem could be solved. The information obtained during the solution of the set covering problem could be used to generate new pairings. These could be inserted into the existing Set Covering Problem, which then could be reoptimized. This would result in an alternating process between generation and selection. Here "morphs" as described in [1] might be usable.

Thirdly, explorations of a traditional mathematical formulation and optimization using a linear optimizer might prove valuable. Even though such an approach is expected to result in very long running time for larger problems, it might be used as a reference when evaluating the developed heuristics.

As a final remark, an important related problem came up during this project; during the tracking phase SAS buys sparetime from crew to close open flights. However, it is not straightforward to decide when to buy the sparetime, and how much to buy from which crew. Due to the dynamic nature of the problem closed flights might become open or open flights might become closed. As operation approaches buying sparetime from crew becomes more and more expensive, making it desirable to buy sparetime at an early point in time. However, as operation approaches more and more information becomes available (passenger forecast might decrease thereby lowering the number of crew needed on a flight) which might reduce the number of crew needed. Thus making it possible to buy less sparetime, however, at a higher price. Hence the problem of deciding when to buy sparetime and how much to buy is also important when running an airline company.

Bibliography

[1] M. Brusco, L. Jacobs, and G. Thompson. A morphing procedure t supplement a simulated annealing heuristic for cost- and coverag correlated set-covering problems. *Annals of Operations Researc* 86:611–27, 1999.

[2] A. Caprara, P. Toth, and M. Fischetti. Algorithms for the set coverin problem. *Annals of Operations Research*, 98:353–71, 2000.

[3] H. Dawid, J. Konig, and C. Strauss. An enhanced rostering mod for airline crews. *Computers and Operations Research*, 28(7):671–68 2001.

[4] G. Desaulniers, J. Desrosiers, Y. Dumas, S. Marc, B. Rioux, M. M Solomon, and F. soumis. Crew pairing at air france. *European Journ of Operational Research*, pages 245–259, 1997.

[5] R. J. E. Gamma, R. Helm and J. Vlissides. *Design Patterns*. Addiso Wesley, 1995.

[6] G. W. Graves, R. D. McBride, I. Gershkoff, D. Anderson, an D. Mahidhara. Flight crew scheduling. *Management Sceinc* 39(6):736–745, June 1993.

[7] L. Lettovsky, E. Johnson, and G. Nemhauser. Airline crew recover *Transportation Science*, 34(4):337–48, 2000.

[8] D. Levine. Application of a hybrid genetic algorithm to airline cre scheduling. *Computers and Operations Research*, 23(6):547–558, 199

[9] P. Lucic. Simulated annealing for the multi-objective aircrew roste ing problem. *Transportation Research Part A: Policy and Practic* 33(1):19–45, 1999.

[10] Z. Michalewicz and D. B. Fogel. *How to Solve It: Modern Heuristic* Springer, 2000.

[11] Analyse af yderligere omdisponeringsmuligheder i TAP-AI, august 199 Revision: 1.02.

[12] C. R. Reeves. Genetic algorithms for the operations researcher. *INFORMS*, 9(3):231–249, 1997.

[13] M. Stojkovic, F. Soumis, and J. Desrosiers. The operational airline crew scheduling problem. *Transportation Science*, 32(3):232–45, 1998.

[14] P. Vance, C. Barnhart, E. Johnson, and G. Nemhauser. Airline crew scheduling: a new formulation and decomposition algorithm. *Operations Research*, 45(2):188–200, 1997.

[15] G. Wei, G. Wu, and M. Song. Optimization model and algorithm for crew management during airline irregular operations. *Journal of Combinatorial Optimization*, 1(3):305–21, 1997.