CHALMERS
UNIVERSITY OF TECHNOLOGY

UNIVERSITY OF GOTHENBURG

Creating Initial Solutions for the Tail Assignment Problem

Master's thesis in Computer Science — Algorithms, Languages and Logic

ELIN BLOMGREN

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2018

Master's thesis 2018

Creating Initial Solutions for the Tail Assignment Problem

ELIN BLOMGREN

Department of Computer Science and Engineering
Chalmers University of Technology
University of Gothenburg
Gothenburg, Sweden 2018

Creating Initial Solutions for the Tail Assignment Problem
ELIN BLOMGREN

Supervisor: Birgit Grohe, Department of Computer Science and Engineering
Supervisor: Ann-Brith Strömberg, Department of Mathematical Sciences
Advisor: Viktor Almqvist, Jeppesen
Examiner: Devdatt Dubhashi, Department of Computer Science and Engineering

Master's Thesis 2018
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Typeset in LaTeX
Gothenburg, Sweden 2018

Creating Initial Solutions for the Tail Assignment Problem
ELIN BLOMGREN
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg

Abstract

One of many optimization problems in the airline industry is the tail assignment problem, i.e. to decide which aircraft operate which flight. Initial solutions can be used to warm-start optimization algorithms. In this thesis, the optimization algorithm uses a time window heuristic together with column generation. This thesis investigates different methods to create initial solutions for the tail assignment problem. The chosen methods consist of greedy algorithms and other simple heuristics. The goal was that the methods should assign at least 95% of the flights and this is achieved by most methods and test cases. Also, when using the produced initial solutions as input to the optimization algorithm, the value of the objective function is improved for some of the test cases.

Keywords: airline optimization, heuristics, greedy, initial solution, tail assignment, aircraft routing, column generation, warm-start of algorithms.

Acknowledgements

Firstly, I want to thank Jeppesen for letting me do my thesis at their office and for providing me with the necessary resources. I especially want to thank my supervisor Viktor Almqvist for the meaningful discussions about my work and for always taking the time to help me. I also want to thank Mattias Grönkvist who has supported me during the thesis.

I want to thank the optimization teams as well as everyone else at the Jeppesen office who have shown interest in the project and to my roommate at Jeppesen, Emily Curry, for keeping me company and cheering me up when the thesis work was hard.

At Chalmers, I want to thank my examiner Devdatt Dubhashi and my two supervisors Birgit Grohe and Ann-Brith Strömberg for their feedback on my thesis work and especially on the writing of the report.

Elin Blomgren, Gothenburg, June 2018

Glossary

Task	A flight, a maintenance, a sequence of tasks or some other activity that should be assigned to an aircraft
Roster	A sequence of tasks that is assigned to an aircraft
Pre-assigned task	A task that is locked to a specific aircraft
Free task	A task that is not pre-assigned to any aircraft

Acronyms

AFR	Aircraft First Random method
AFS	Aircraft First Sort method
AFM	Aircraft First Maintenance method
TFR	Task First Random method
TFS	Task First Sort method
TFM	Task First Maintenance method
IMP	IMProvement method
DFS	Depth First Search
LP	Linear Program
ILP	Integer Linear Program
MP	Master Problem
RMP	Restricted Master Problem

Notation

A	The set of all aircraft
C	The set of all cumulative rules
$f \in F$	A task in the set of all tasks
$r \in R$	The roster in the set of all rosters
$F_r \subseteq F$	The set of tasks in roster r
$a_r \in A$	The aircraft assigned to roster r
t_f^{start}	The starting time of task f
t_f^{end}	The ending time of task f
p_f^{start}	The starting position of task f
p_f^{end}	The ending position of task f

Contents

1

Introduction

There are many decisions to be made at an airline. One important problem is to decide which aircraft should operate which flight. To do this, one has to consider operational constraints such as maintenance, airport curfews, and aircraft restrictions. This problem is called tail assignment [1]. In addition to satisfying all rules and constraints, the problem can be modeled to minimize fuel consumption, maximize aircraft utilization, increase robustness, etc., in order to find savings and increase the reliability of the airline's operation.

This thesis is done in collaboration with Jeppesen in Gothenburg. Jeppesen is a subsidiary of The Boeing Company and provides planning and optimization tools for airlines. One of their products is a tail assignment optimizer which is the starting point for this thesis.

Aircraft operating costs are the by far biggest operating cost for airlines [2, Chapter 6]. The single largest cost in this category is fuel. Further, the total cost of delayed flights in US in 2007 was estimated to be $31.2 billion [2, Chapter 10]. Making robust schedules that minimize the delays and their consequences is therefore an important way to find savings for airlines. Different airlines prioritize costs and robustness differently. By modeling robustness and other qualities as costs, Jeppesen's products allow the airlines to choose which qualities that are most important and the optimizer finds a solution targeted to minimize the cost.

The tail assignment problem is an NP-hard optimization problem [1, Section 4.2], which means that it can sometimes be very computationally demanding to find good solutions. To find good enough solutions as quickly as possible, Jeppesen's optimizer uses a hybrid column generation and local search solution approach. The problem's planning period is divided into time windows, and the problem is solved using column generation in a sequence of time windows which 'slide' over the planning period to gradually cover the full period.

To improve the performance of the currently in use optimizer at Jeppesen, an initial solution that covers the entire planning period can be used as input to the optimizer. Producing solutions fast is also good because the user of the system gets feedback almost immediately and the user can also see if the problem data and rules work correctly. This thesis will investigate methods to create initial solutions and evaluate them on real problems from different airlines.

1.1 Aim and Limitations

The goal of this thesis is to find and implement a method that can be used to create initial solutions to warm start an improvement method. Since it can be hard to assign all flights for complex problem instances, the goal is that the initial solutions should have at least 95% of the flights assigned. All other hard constraints (explained in Section 3.1.2) must be satisfied. The aim is that the current optimizer at Jeppesen can produce better solutions if given an initial solution produced with the developed method, than without an initial solution. The method also needs to be fast and scalable to be able to support large airlines.

The main limitation of this thesis is that the methods developed in this thesis do not aim at finding the optimal solution. Since the produced solutions will be used as initial solutions to an improvement method, it is enough that it is a legal solution covering the whole planning period (but with some flights unassigned).

The testing will be limited to a few problem instances from Jeppesen's test suite. The final aim is that the method should work for all possible instances, but for the scope of this thesis there will be a smaller selection of problem instances for the evaluation.

1.2 Literature Review

Grönkvist [1] presented a constraint programming approach as well as a column generation approach to solve the tail assignment problem. Later, Gabteni and Grönkvist [3] combined these approaches to quickly find initial solutions as well as to improve the solution quality. Two difficulties with their approach are that it does not work if there is no feasible solution and that it does not consider maintenance or other cumulative constraints.

One of the most recent publications on tail assignment is Khaled et al. [4]; it presents a compact model of tail assignment which is solved to optimality. The bigger instances are, however, not solved to optimality when the maintenance constraints are added. There is only one type of maintenance considered in [4]. and their biggest instance only has 40 aircraft and 1494 flights.

Sarac et al. [5] present a branch-and-price method to the operational aircraft maintenance routing problem. A problem that is similar to tail assignment, but with the main difference that the planning horizon is shorter. In [5] only one day was used and their objective was to minimize the number of unused legal flying hours. However, the first steps in their method for creating initial solutions are similar to the ones presented in this thesis in the way that the aircraft are sorted and that the first possible connection is chosen iteratively.

Another solution method for aircraft maintenance routing proposed by Safaei and

Jardine [6] includes a connection network Integer Linear Program (ILP) that handles various maintenance tasks. They only solve the problem for one week at a time for a fleet with up to 18 aircraft. Another difference is that they allow non-revenue flights to transport aircraft to maintenance locations.

Column generation is a frequently used approach for many airline problems [1], [5], [7], [8]. However, Amin Jamili [9] and Deng and Lin [10] are two examples of the use of stochastic optimization algorithms [11].

A similar problem to tail assignment is railway rolling stock assignment, the problem of assigning train-sets to utilization paths. Lai et al. [12] focused on the maintenance for rolling stock and showed good results with a heuristic approach. Among other techniques, their algorithm start with the trains that need maintenance the soonest. However, their routes are already aggregated into trips that all begin and end at the same location which makes the problem easier to solve.

Creating initial feasible solutions for different operations research problems is often a domain specific task. Several articles on initial solutions are published in different domains. For example, Juman and Hoque [13] for the transportation problem and Joubert and Claasen [14] for the constrained vehicle routing problem. Also, Guedes and Borenstein [15] show that a good initial solution improved the running times significantly for the multiple-depot vehicle type scheduling problem.

Tail assignment can be modeled as an integer multi-commodity network flow problem with resource constraints [1, Model 4.1]. Dai et al. [16] proposed several approaches to initial solutions for the multi commodity network flow problem. These will, however, not work very well in the tail assignment setting since the authors in [16] focus on problem instances with more commodities than nodes, which would correspond to more aircraft than flights. Also, the resource constraints and the integrality constraints would have to be relaxed.

1.3 Thesis Outline

In Chapter 2 the theory used for both Jeppesen's current solver as well as for the methods proposed in this thesis is explained. After that follows a more in-depth problem definition in Chapter 3. In Chapter 4, the methods developed in this thesis are presented, followed by the tests and results in Chapter 5. Finally, the thesis is concluded in Chapter 6.

2

Theory

This chapter describes the theory used for different solution methods for the tail assignment problem. Section 2.1 describes a common way to model integer linear optimization problems and Section 2.2 describes one way to solve these optimization problems. These techniques are currently in use at Jeppesen. Section 2.3 describes the theory behind the main algorithm proposed in this thesis.

2.1 Integer Linear Programs

Linear Programs (LP) are a way to model optimization problems with continuous variables, linear constraints and linear objective function [17, Chapter 4.3]. Let x be a vector of n variables and c be a vector with corresponding costs for each variable. The m number of linear constraints for the variables x are expressed in the constraint matrix $A \in \mathbb{R}^{m \times n}$ together with the vector $b \in \mathbb{R}^m$ with the bound for each constraint. Then a standard LP is of the form

$$\begin{aligned} \min \quad & c^\intercal x, && \text{(2.1a)} \\ \text{s.t.} \quad & Ax = b, && \text{(2.1b)} \\ & x \geq 0. && \text{(2.1c)} \end{aligned}$$

The goal is to minimize the cost (2.1a) subject to a set of linear equality (or inequality) constraints (2.1b), where constraint i is defined as $A_i x = b_i$.

One of the most widely used methods to solve LPs is the Simplex method [18]. Even though the worst case complexity of the Simplex method is exponential in theory, it has been proved that the so called smoothed complexity is polynomial [19], [20]. Examples of other solution methods are the ellipsoid method and interior point methods [21].

Integer Linear Programs (ILP) are special kinds of LPs where the variables must have integer values. Therefore, there is also a constraint on the form $x \in X$, where

$X \subseteq \mathbb{Z}^n$. For a binary ILP, $X = \{0,1\}^n$ and a standard binary ILP is of the form

$$\begin{aligned} \min \quad & c^\intercal x, && \text{(2.2a)} \\ \text{s.t.} \quad & Ax = b, && \text{(2.2b)} \\ & x \in \{0,1\}^n. && \text{(2.2c)} \end{aligned}$$

In contrast to LP, which can be solved efficiently, ILPs are generally NP-hard and no efficient algorithm is known [22]. An ILP can be relaxed to allow continuous values of the variables. Such a re-formulated problem is called an LP-relaxation of the ILP. The LP-relaxation can be solved efficiently (e.g. by the Simplex method) but this will of course in general produce continuous solutions that is not feasible for the ILP.

One common exact method to solve ILP is Branch-and-bound [23, Section 8] where relaxed problems are solved repeatedly in a tree-structure. In each node, the feasible region is divided into two parts, which becomes two branches. Both the lower and upper bound are stored in each node and a branch is pruned if it is not possible (calculated by the bounds) that the optimal solution can be found in a sub-branch of the node.

2.2 Column Generation

Column generation is a method that can be used to solve large scale LPs and ILPs [24]. To apply column generation, it is necessary to first reformulate the problem into two problems. These are a master problem (MP) and one or more sub problems (also called pricing problems). For example, the resource constrained shortest path problem [25] can be modeled with a decision variable for each edge. When reformulated, a decision variable in the MP will instead represent a possible path through the network. Each decision variable in the MP is associated with a path through the network and is represented by a cost coefficient in the objective and a column in the constraint matrix. The number of paths is typically very large and therefore only a subset of the paths are generated. The sub-problems are then used to generate new possible paths (columns) with negative reduced cost (i.e. columns that can improve the objective) to the restricted master problem (RMP). This is repeated for a predefined number of iterations or until the sub-problems do not find any improving columns.

During the iterations, the RMP passes new (dual) information to the sub-problems and the sub-problems passes new improving columns to the RMP. But to initialize the iterations, a set of initial feasible columns is needed [26]. These can be defined using so-called artificial variables [27], one for each constraint, each with a high cost, or, one can find initial solutions in other ways.

The RMP is solved as an LP. If the actual problem is an ILP one has to use some fixing heuristic to find an integer solution. Barnhart et al. [28] combine column

generation with branch-and-bound to solve large-scale ILP and the resulting method is called branch-and-price.

2.3 Greedy Algorithms

Greedy algorithms always make the choice that is best (according to some criteria) at the moment [29]. In general, greedy algorithms are not guaranteed to find an optimal solution, but for some problems there exists greedy algorithms that always find optimal solutions. For other problems, greedy algorithms can be used to find good feasible solutions.

Generally, one can define a greedy algorithm with three sets and four functions [30]. The starting point is a set C of all possible candidates and two empty sets, one for the solution S and one with discarded candidates D. As the algorithm proceeds, candidates from the set C will be moved to S or D. A selection function decides which candidate $c \in C$ is most promising and a feasibility function checks if a set of candidates can form a solution by adding more candidates. There is also a function that checks if a set of candidates form a complete solution and finally an objective function that gives a solution an objective value. A scheme of a general greedy algorithm is shown in Algorithm 1.

Algorithm 1: Greedy scheme

```
1  S ← ∅
2  D ← ∅
3  while C ≠ ∅ and not isSolution(S) do
4  |  x ← select(C)
5  |  C ← C \ {x}
6  |  if isFeasible(S ∪ {x}) then
7  |  |  S ← S ∪ {x}
8  |  else
9  |  |  D ← D ∪ {x}
10 |  end
11 end
12 if isSolution(S) then
13 |  return (S, objective(S))
14 end
```

The algorithm selects in each iteration the best candidate $c \in C$ according to the selection function. This candidate is removed from C and if it is feasible to add to the solution S, it is added. Otherwise it is discarded and placed in D. After each iteration, the algorithm checks whether or not the set S forms a complete solution. When either the set of candidates is empty or the set S forms a complete solution, the algorithm terminates and returns S as the solution, optionally together with the objective value of the solution.

3

Tail Assignment

Aircraft are identified by their tail-number, which is why the problem studied in this thesis is called tail assignment. Tail assignment is an optimization problem and the goal is to assign aircraft to all flights while optimizing some objective. Section 3.1 describes the tail assignment problem and Section 3.2 describes different models used when solving tail assignment.

One of the main differences between Jeppesen's optimizer and others seen in literature is that Jeppesen's optimizer has a very flexible way to model the constraints and the objective function. Therefore, it is hard to make a specific definition of the components of the problem since these tend to vary between airlines. This chapter aims at describing the components in general.

3.1 Problem Description

Tail assignment is solved for a specific planning period, which means that there is a start and end date. Typically this is about a month but it can be both shorter and longer.

3.1.1 Tasks and Notation

Flights and maintenance checks are modeled as tasks, denoted by f. A task can also be a composite of other tasks. The set F consists of all tasks f that are scheduled in the planning period. Each task $f \in F$ has a start time t_f^{start} and an end time t_f^{end}. Each task f also has a starting position p_f^{start} and an end position p_f^{end}. In principle, a task can be anything possessing these attributes.

A roster is a sequence of tasks that belong to a specific aircraft. Let A be the set of all aircraft and R the set of all rosters. Then, the aircraft assigned to the roster $r \in R$ is denoted by $a_r \in A$. The sequence of tasks belonging to roster r is $F_r \subseteq F$. Since a roster is simply a sequence of tasks and a task can be a composite of tasks, the same notation as for tasks is used for the roster attributes, i.e. t_r^{start} etc.

Some tasks are pre-assigned, which means that they are locked to a specific aircraft. There are two types of pre-assigned tasks. The first type is carry-ins, which are the tasks that are scheduled before the planning period begins and states where the aircraft is in the beginning of the planning period. The second type is pre-scheduled maintenance, typically the bigger ones that lasts for at least one week. Different maintenance types are explained in the next section.

An example solution to tail assignment is illustrated in Figure 3.1. The example consists of four aircraft and a planning horizon of four days. The gray tasks are pre-assigned and locked to the aircraft. The blue tasks are so-called free tasks, for which it is up to the optimizer to decide which aircraft the task should be assigned to. The pre-assigned task at day 3 is an example of pre-assigned maintenance which is described below.

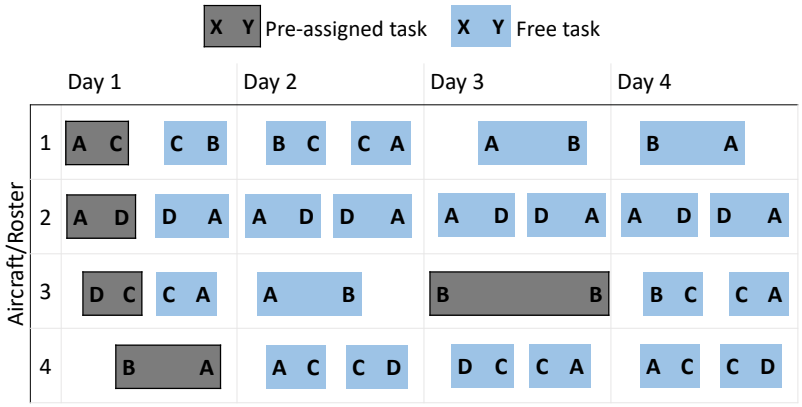


Figure 3.1: An example of a solution to a tail assignment problem. The tasks are boxes with a starting position and an ending position.

Pre-assigned tasks may result in gaps in the roster between the pre-assigned tasks. For example, the roster for aircraft 3 in Figure 3.1 had a gap between the pre-assigned tasks at day 1 and 3, before the free tasks were assigned to the aircraft.

3.1.2 Constraints and Maintenance

At Jeppesen, all rules concerning flights, airports, tasks, aircraft, maintenance etc. are written in Rave [31]. Rave is a domain specific language developed at Jeppesen. The optimizer has access to a general interface to Rave where certain questions can be asked. For example, if a task is legal for a specific aircraft or to get the connection cost between two tasks.

There are different kinds of maintenance for aircraft. The most extensive kind of maintenance is done every one to three years and can take up to a month. The smallest maintenance might be performed every day (night) and may take about an hour. Between these extreme cases, there are a lot of intermediate maintenance levels. The maintenance regulations vary between airlines and countries and multiple maintenance levels can be used by the same airline.

When modeling tail assignment, there are two ways to model the maintenance rules. The first way is as pre-assigned tasks, which means that the maintenance's time and place are not decided by the optimization process but instead decided on beforehand. Usually, it is the bigger and less frequent maintenance that is modeled in this way. The second way is with cumulative constraints, which are usually used for the more frequent maintenance types that are performed several times during the planning period. The same problem instance can have both kinds of rules for different maintenance levels.

Cumulative constraints are the most complex rules since they depend on the history of the aircraft (e.g. it is needed to keep track of when the aircraft last had maintenance of a certain type). Cumulative constraints are modeled as resource constraints where the resource for example can be time, flight hours, or number of landings. If the consumption of the resource exceeds a defined limit, the constraint is violated. The most common way to reset the consumption of the resource is to make sure that the aircraft spends a number of hours on the ground at one in a subset of the airports (e.g. at which it is possible to perform maintenance). The set of cumulative constraints are denoted by C.

There are both soft and hard constraints. The hard constraints simply state that something is impossible and that a solution that breaks any hard constraint is illegal. The soft constraints are modeled with a penalty cost (in the objective) for violating the constraint.

Since tail assignment can be solved simultaneously for multiple fleets, it can sometimes be cost-worthy to re-fleet a flight. Re-fleeting means that a flight is assigned to an aircraft of a different type than it was originally planned for. Re-fleeting can be modeled as a soft constraint where the penalty for violating the constraint is the cost of changing aircraft type.

Sometimes there are also global constraints, which apply when something is dependent on more than one aircraft. The most recurring global constraint is to have a limit on the number of aircraft that receive maintenance simultaneously at the same airport. Another example is to require a minimum number of aircraft on the ground, to use if disturbances occur.

3.1.3 Objective

Different objectives can be used when solving the tail assignment problem. The objective can be to minimize actual costs, such as fuel consumption, or fictive costs, which for example will favor robust solutions or other qualities that the airline desires. Often both actual and fictive costs are used in combination.

The objective function is also modeled in Rave and is to be defined by the user of the system. If the problem instance does not have a solution covering all flights, the cost of unassigned tasks is often the biggest part of the total cost. Another part

that often influences the cost is fuel consumption and the connection time between flights. An example of a connection cost function that can be used to reduce medium length connections is found in [1, Section 13.1]. When soft constraints are employed, the penalty for violating these are also part of the cost.

3.2 Optimization Models

There are multiple ways to model the tail assignment problem. Grönkvist [1] presents three main models. First, there is an integer multi-commodity network flow problem formulation [1, Model 4.1]. This is perhaps the most intuitive formulation but it is not used very often when solving the tail assignment problem. Further, there is a set partitioning formulation [1, Model 4.2]. This is a very useful model that can be solved by column generation and is explained below. The last model is a constraint satisfaction problem model [1, Model 9.2] that Grönkvist later uses for accelerating the column generation [32]. This can be done since all three of these models express the same problem in different ways and a solution to one formulation can always be transformed into a solution to the others.

Let R be the set of all legal aircraft rosters, x_r be the decision variable of whether or not roster $r \in R$ is taken and c_r be the cost of roster r. Further, the constant α_{fr} is 1 if task f is covered by roster r and 0 otherwise. Then, tail assignment can be modeled as a set partitioning problem [1, Model 4.2]

$$\min \quad \sum_{r \in R} c_r x_r, \tag{3.1a}$$

$$\text{s.t.} \quad \sum_{r \in R} \alpha_{fr} x_r = 1, \qquad \forall f \in F, \tag{3.1b}$$

$$x_r \in \{0, 1\}, \qquad \forall r \in R. \tag{3.1c}$$

The goal as specified in (3.1a) is to choose $x_r, r \in R$ such that the total cost is minimized. In this problem formulation there are two sets of constraints. In (3.1b) we ensure that each task is covered by a an aircraft and the integrality constraints in (3.1c) ensure that x_r is binary.

The complicating property of this formulation is that the set R is very large (exponential in the number of tasks) and also hard to find [1, Section 4.4 and 5.2]. The current approach to solve this is to use column generation (see Section 2.2). The RMP is the LP-relaxed (i.e. (3.1c) changed to $x_r \geq 0$) version of (3.1) but with the subset $R' \subseteq R$ instead of R. The pricing problem is then used to generate new rosters to R' (i.e., new columns, consisting of α_{fr}, $f \in F$, and c_r). This problem is modeled in a graph as a resource constrained shortest path problem [1, Section 5.3] to ensure that the generated rosters satisfy all the constraints. Some constraints are enforced by the connections between nodes. The cumulative constraints (e.g. maintenance) are modeled with the resources in the pricing problem. For more details, see [1, Section 5.3–5.4].

3.3 Solving Tail Assignment Using Time Windows

At Jeppesen, most problem instances are divided into multiple smaller so-called time windows for which column generation is performed individually [1, Section 12.3]. Usually the size of the windows are between one and seven days. One sweep through the planning period goes through all time windows once in order. The time windows have some overlap that is decided per customer. After one sweep, the time windows can be altered so that the breakpoints occurs at different places in the next sweep.

For each time window, there are some locked tasks in the beginning and in the end. The rosters generated in the column generation only cover this time window and consists of a path between the start task and end task. If there is no initial solution, each time window only has one fixed task in the beginning, the end task is up to the column generation to decide.

4

Heuristics to Create Initial Solutions

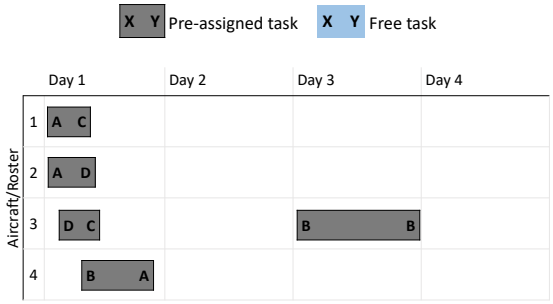
This chapter presents the developed methods to create initial solutions to the tail assignment problem. The methods use heuristics to find solutions fast. The heuristics used are greedy algorithms and depth first search (DFS) [33, Section 3.2].

The general method is divided into two stages. First, the gaps between pre-assigned tasks are handled using a method called `FillGaps` and then, in the second stage, the rosters are appended with more tasks using the method `AppendRosters`. An example that illustrates these stages are found in Figure 4.1. This means that `AppendRosters` will assign tasks (if possible) after the last pre-assignment in each roster. The general algorithm is presented in Algorithm 2.

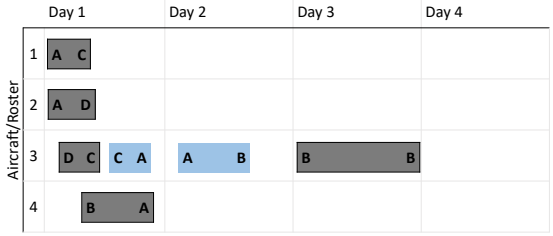
Algorithm 2: General scheme

1 `Initialize` A, F, R
2 `shuffle`(R)
3 `FillGaps`(R, F)
4 `AppendRosters`(R, F)

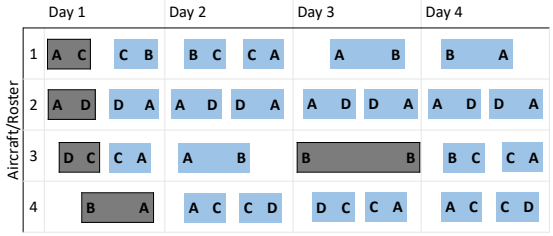
The first row in Algorithm 2 is the initialization steps. First, the data (tasks F and aircraft A) is collected. Then, for each aircraft $a \in A$, a new roster r is created with only the pre-assigned tasks for a_r. The rosters form the set R. The set F is ordered by ascending start time. After that, the rosters are shuffled to avoid getting the same arbitrary order for each run of the algorithm. In this way, we can run the algorithm multiple times and then choose the best solution. Row 3 handles the gaps between pre-assigned tasks and is explained in detail in Section 4.2. Finally, the method `AppendRosters` on row 4 is done in several different ways, which are described in Sections 4.3 and 4.4. Section 4.1 describes how to determine whether a connection is legal, something that is used in many of the developed methods.



(a) The initial four rosters with only pre-assigned tasks.



(b) The four rosters resulting from the application of the method `FillGaps`.



(c) The four rosters resulting from the application of the method `AppendRosters` has been performed.

Figure 4.1: An example of how a solution is formed by the stages in the general scheme.

4.1 Legal Connections

In many situations, there is a need to check if the connection between two tasks is legal. To see if the connection is legal, there are a number of conditions to check. We define the function `isLegal`(f, g, a) to check the legality of the connection between tasks f and g with respect to aircraft a. Note that the first task can be either a roster or a task. The following legality checks are performed:

1. Is the position $p_f^{\text{end}} = p_g^{\text{start}}$?
2. Is the time $t_f^{\text{end}} \leq t_g^{start}$?
3. Does aircraft a has the necessary requirements needed for task g?

4. Is task g is allowed for aircraft a according to the Rave model?

5. Is the connection between task f and g legal for aircraft a according to the Rave model?

For a connection to be legal, all these checks must be true. The checks are performed in increasing order, so that the more time-consuming checks (which depend on the Rave model) are only done if the previous checks were legal.

For check number 2, we assume a connection time of 0. In practice, a longer connection time is needed, but since this depends on the task, the airport, and the aircraft, this is handled in Rave (check 5).

4.2 Fill Gaps Between Pre-assigned Tasks

Pre-scheduled maintenance create gaps in the rosters between the pre-assigned tasks in the beginning of the planning period and the later maintenance. This is one of the more difficult parts for the current optimizer. Therefore, it is important to find an algorithm that can fill these gaps efficiently.

The method `FillGaps` is shown in Algorithm 3. The algorithm loops through the tasks in each roster. The next task after f is denoted $f+1$. For each task f in the roster r, a DFS inspired algorithm is used to find a sequence of tasks that fill the gap between tasks f and $f+1$. The tasks represent the nodes in the graph searched by the DFS. The only difference from an ordinary DFS is that it only checks if the goal is reached if there are no more possible tasks to add. The set V of visited nodes (tasks) is used to avoid visiting the same node more than once.

For each gap in the roster at hand, the algorithm will check all tasks $g \in F$, for which t_g^{start} is between t_f^{end} and t_{f+1}^{start}, whether it is legal to assign task g after task f. The algorithm also check that the task is uncovered and whether the inequality $t_g^{\text{end}} \leq t_{f+1}^{\text{start}}$ holds. If legal, the task g is appended to the roster r. When there are no more tasks to consider, the algorithm checks if the new connection in the roster, i.e. between f and $f+1$, is legal. If it is not, the task f is removed from the roster and put in the set of visited nodes $V \leftarrow V \cup \{f\}$. Let f be the previous task in the roster and go through the steps again.

If there are two pre-assigned tasks for each roster, one in the very beginning of the planning period and one in the very end, there are $|A|$ gaps to fill with $|F| - |A|$ potential tasks. These are considered by DFS, for which the time complexity is $O(|\text{nodes}| + |\text{edges}|)$; the nodes correspond to the potential tasks $|F| - |A|$, and there are at most $(|F| - |A|)^2$ edges. The resulting complexity is $O(|A|(|F| - |A|)^2) = O(|A||F|^2 - 2|F||A|^2 + |A|^3)$. Since $|A| \ll |F|$, this means that $|A|^3 < |A||F|^2$ and the complexity can be written as $O(|A||F|^2)$. Note that this is a worst case analysis and this will probably not be the case for real problems.

Algorithm 3: `FillGaps`

```
1  foreach r ∈ R do
2      f ←first(r)
3      V ← ∅
4      while f ≠ null do
5          foreach g ∈ {g : g ∈ F ∧ t_f^end ≤ t_g^start ≤ t_{f+1}^start} do
6              if isLegal(f, g, a_r) ∧ g ∉ V ∧ t_g^end ≤ t_{f+1}^start ∧ isUncovered(g) then
7                  insert(r, g)
8                  f ← g
9              end
10         end
11         if isLegal(f, f + 1, a_r) then
12             f ← f + 1
13         else
14             remove(r, f)
15             V ← V ∪ {f}
16             f ← f − 1
17         end
18     end
19 end
```

4.3 Greedy methods

The methods to be presented in this section do not consider hard cumulative constraints but only the ones checked in the function `isLegal`. The reason for this is to be able to use a fast greedy approach for the problem instances that does not have hard cumulative constraints (see Section 4.4 for the corresponding algorithms for those cases). The four methods are all quite similar. These are Aircraft First Random (AFR), Aircraft First Sort (AFS), Task First Sort (TFS) and Task First Random (TFR). For the complete run of Algorithm 2, one of these is chosen as the method `AppendRosters`.

Recall the notation used for the general greedy algorithm in Section 2.3. The candidate set C is here the set of pairs $(a, f) \in A \times F$ (the Cartesian product of aircraft and tasks), the solution set S corresponds to R, in which the rosters are accumulated, and the set D of discarded candidates is accumulated implicitly by the way new candidates are selected.

The selection function is implicit by the looping through the set of candidates. The looping order differs between the methods. The feasibility function is the function `isLegal` and there is no explicit function checking if the solution is complete, since all the candidates will be considered. Finally, the objective function is the one defined in the Rave model (as explained in Section 3.1.3).

4.3.1 Aircraft First

Each of the two methods AFR and AFS consists of a nested loop where the outer loop iterates through the aircraft/rosters and the inner loop iterates through the tasks.

Before these loops are executed, the aircraft are ordered in some way. For AFR, the aircraft are listed in random order and is therefore called Aircraft First Random (AFR), see Algorithm 4.

Algorithm 4: Aircraft First Random (AFR)

```
1 shuffle(R)
2 foreach r ∈ R do
3 |   foreach f ∈ F do
4 |   |   if isLegal(r, f, a_r) ∧ isUncovered(f) then
5 |   |   |   append(r, f)
6 |   |   end
7 |   end
8 end
```

The other method AFS sorts the aircraft based on the number of restrictions (decreasing) which is called Aircraft First Sort (AFS). Note that the solution produced by the method can differ between runs if multiple aircraft have the same number of restrictions, since ties are broken randomly.

Algorithm 5: Aircraft First Sort (AFS)

```
1 sort(R)
2 foreach r ∈ R do
3 |   foreach f ∈ F do
4 |   |   if isLegal(r, f, a_r) ∧ isUncovered(f) then
5 |   |   |   append(r, f)
6 |   |   end
7 |   end
8 end
```

The shuffle step in AFR takes linear time ($O(|A|)$) and is done only once in the beginning. Since the outer loop goes through each roster once, and there is one roster for each aircraft, this loop consists of $|A|$ iterations. The inner loop consists of $|F|$ iterations, where each iteration takes constant time. Hence, the resulting time complexity is $O(|A||F|)$.

For AFS, the algorithm starts by sorting the aircraft/rosters which is done in $O(|A|\log(|A|))$. However, to obtain the number of restrictions used for the sorting, $O(|A||F|)$ computations are required. The resulting complexity is then $O(|A|\log(|A|)+|A||F|)$. For practical problems, there are a lot more tasks than aircraft (i.e.

$|A| \ll |F|$). Therefore, the first term can be ignored and the complexity is the same as for AFR, i.e. $O(|A||F|)$.

4.3.2 Tasks First

The Aircraft First methods typically assign a lot more tasks to the first rosters compared to the later rosters. Therefore, we present two methods that aim to distribute the tasks more evenly.

The difference between the Tasks First and Aircraft First methods is that the inner and outer loop have been swapped. Instead of looping over the rosters and for each roster loop over all tasks, the algorithms loop over all tasks once and for each task over all the rosters. In this way, the inner loop may be exited when a legal assignment is found since only one aircraft should cover each task. To distribute the tasks evenly, the order of the rosters are shuffled for each task; see Algorithm 6.

Algorithm 6: Tasks First Random (TFR)

```
1 foreach f ∈ F do
2     shuffle(R)
3     foreach r ∈ R do
4         if isLegal(r, f, a_r) ∧ isUncovered(f) then
5             append(r, f)
6             break
7         end
8     end
9 end
```

An alternative is to re-sort the aircraft for each task. In this case, the sorting is based on the number of tasks (ascending) in each roster to even better balance the load on each aircraft. This is shown in Algorithm 7; note that the only difference from Algorithm 6 is on row 2.

Algorithm 7: Tasks First Sort (TFS)

```
1 foreach f ∈ F do
2     sort(R)
3     foreach r ∈ R do
4         if isLegal(r, f, a_r) ∧ isUncovered(f) then
5             append(r, f)
6             break
7         end
8     end
9 end
```

For TFR, the outer loop consists of $|F|$ iterations and in each iteration, the shuffle step takes $O(|A|)$ time and the inner loop contains at most $|A|$ iterations. Hence, the time complexity is $O(|A||F|)$, the same as for AFR and AFS. For TFS, on the other hand, the sorting is done in $O(|A|\log(|A|))$ and the total complexity is $O(|A||F|\log(|A|))$. As mentioned in the previous section, $|A| \ll |F|$ and this extra $\log(|A|)$ factor should not influence the practical running time much.

4.4 Methods for Hard Cumulative Constraints

When the problem instance possesses hard cumulative constraints, the greedy algorithms described in the previous section will most often generate illegal solutions. We present two methods that can handle these constraints: Aircraft First Maintenance (AFM) and Task First Maintenance (TFM). For problem instances with hard cumulative constraints, one of AFM and TFM is chosen as the method `AppendRosters` in Algorithm 2.

To check if a roster r violates any of the cumulative constraints in the set C, the whole roster has to be considered. This takes at most $O(|C||F_r|)$ time, where $|F_r|$ denotes the number of tasks in the roster r.

4.4.1 Aircraft First with Hard Cumulative Constraints

AFM is inspired by the method `FillGaps` and uses a similar DFS algorithm. However, the goal here is to find a path that is as long as the planning period and therefore a heuristic parameter $T_{\max}$ is introduced.

The general idea is that legal tasks are appended as long as they do not violate any hard cumulative constraint. If a constraint becomes violated by a task, it is not added. The parameter $T_{\max}$ is used to determine (heuristically) whether it is likely that the roster can further be appended. If the time between the end time t_r^{end} of the roster and the start time t_f^{start} of the potential task f to add exceeds the time $T_{\max}$ (i.e. $t_f^{\text{start}} - t_r^{\text{end}} > T_{\max}$), it is considered unlikely that a legal task to append to the roster will ever be found. Therefore, the last task in r is removed and set the task f to the task closest to t_r^{end} and try again to append the roster. All tasks that have ever been in the current roster are forbidden to be added again. This is kept track of this using the set V of visited tasks. See Algorithm 8.

In the worst case, the algorithm exceeds the maximum time $T_{\max}$ once for each task, and all tasks are added at some point, but all are also removed. Therefore, the check for violation of cumulative constraints may be performed at most $|A||F|^2$ times and each check has a worst case of $O(|C||F|)$ computations since the roster can consist of at most $|F|$ tasks. Hence, the complexity of Algorithm 8 becomes $O(|C||A||F|^3)$. This is a lot higher than for AFR, AFS, TFR, and TFS but since this is a worst case analysis, in practice, no rosters consist of $|F|$ tasks, and since the cumulative

Algorithm 8: Aircraft First Maintenance (AFM)

```
1  foreach r ∈ R do
2      V ← ∅
3      f ← first(F)
4      while f ≠ null do
5          if t_f^start − t_r^end > T_max then
6              removeLast(r)
7              f ← min f ∈ F : t_f^start ≤ t_r^end
8          if isLegal(r, f, a_r) ∧f ∉ V∧ isUncovered(f) then
9              if isCumulativeLegal(r, f, a_r) then
10                 append(r, f)
11                 V ← V ∪ f
12             end
13         end
14         f ←next(F, f)
15     end
16 end
```

constraints are only evaluated if the connection is legal in all other aspects, the algorithm will be faster in most practical settings.

AFM provides a simple way to handle all kinds of possible cumulative constraints. It does not matter what resources are consumed in which tasks and how the resource is reset for each constraint (see Section 3.1.2). The only assumption is the maximum time $T_{\max}$, which is a heuristic parameter whose value must be higher than the maximum reset time of any hard cumulative constraint and lower than an acceptable connection time between tasks. Because of the way tail assignment is modeled at Jeppesen, such detailed information about rules is not available in the interface to the rule model. Instead, $T_{\max}$ is manually set to one day.

4.4.2 Task First with Hard Cumulative Constraints

The method TFM is a naive approach that uses the same algorithm as TFR but in addition to the `isLegal` check, it also checks the hard cumulative constraints; see Algorithm 9. This approach does not always create rosters that cover the whole month, since the aircraft might be stuck at an airport where maintenance is not possible to perform.

For the complexity analysis, the worst case occurs when all tasks are possible to assign to the last roster that is tried. As previously stated, the worst case for the check for cumulative constraints violation has a complexity of $O(|C||F|)$. This is tried at most $|F||A|$ times and the resulting time complexity is $O(|C||A||F|^2)$.

Algorithm 9: Tasks First Maintenance (TFM)

```
1  foreach f ∈ F do
2      shuffle(R)
3      foreach r ∈ R do
4          if isLegal(r, f, a_r) ∧ isUncovered(f) then
5              if isCumulativeLegal(r, f, a_r) then
6                  append(r, f)
7                  break
8              end
9          end
10     end
11 end
```

5

Tests and Results

The methods described in Chapter 4 have been implemented in C++ in Jeppesen's development environment. The testing has been done on a machine with Intel(R) Xeon(R) CPU E5-2667 v2 @ 3.30GHz. It has two times eight cores but only 8 are used per optimization run.

The different methods in Chapter 4 are evaluated on the test cases presented in Table 5.1. These differ in size in several ways. Cases 3 and 4 have the shortest planning period, of one week, while the others have a planning period of just over a month. They also differ in number of aircraft. Case 8 includes seven aircraft while case 5 includes 112 aircraft. Case 5 is also the largest in terms of number of tasks.

The first six cases have none or only soft cumulative constraints while cases 7 and 8 have hard cumulative constraints.

The test cases have other differing characteristics, not shown in the table. For example, they have different rules, pre-assignments, and airports. They also have different objective functions. Case 2 possesses only the unassigned task cost while cases 3, 4, and 8 also have connection costs. The cases 1, 5, 6, and 7 have these costs as well as penalties for violating soft cumulative constraints.

	#aircraft	#tasks	#days	UB	LB
Case 1	15	2231	37	424 904	205
Case 2	46	7730	36	740 775	0
Case 3	71	3096	7	30 254	1 344
Case 4	17	727	7	70 822	1 656
Case 5	112	23825	32	2 372 428	64
Case 6	65	6191	36	121 636	51
Case 7	27	4388	34	16 407	766
Case 8	7	1151	39	138 885	4 801

Table 5.1: Test cases with some basic characteristics. The columns represent the number of aircraft, the number of tasks, the length of the planning period, the upper bound (UB) (the cost of the initial solution with only pre-assigned tasks) and a lower bound (LB) (see description in the text).

The lower bounds (LB) presented in Table 5.1 are calculated using a network flow

model according to [1, Section 4.6]. This lower bound considers neither aircraft specific connection rules nor aircraft specific costs. If everything can be assigned without these aircraft specific rules, the only cost in the lower bound will be the connection cost since penalties for soft constraints (cumulative or global) cannot be handled efficiently in the network flow model. For all the test cases in Table 5.1 except case 7, the lower bound solutions have all tasks assigned. Therefore, the lower bound consists of the connection cost only. The quality of the lower bound differs a lot between the test cases. For example, the lower bound for case 5 is much lower than the optimal solution since the costs differ between aircraft and since no penalties are included.

Our main focus has been on comparing the methods with each other but also with the improvement method (IMP) developed by Jeppesen. IMP goes through the planning period once, divided in time windows (as explained in Section 3.3). The size of the time windows varies between test cases but is the same for a single case when run with and without an initial solution.

Since all the methods described in Chapter 4 have some random components, each method has been run 49 times each per test case in the evaluation. The greedy methods Aircraft First Random (AFR), Aircraft First Sort (AFS), Task First Sort (TFS), and Task First Random (TFR) are evaluated in Section 5.1 on test cases 1–6. The methods Task First Maintenance (TFM) and Aircraft First Maintenance (AFM) are evaluated in Section 5.2 on test cases 7 and 8.

5.1 Results for Test Cases Without Hard Cumulative Constraints

The percentages of assigned tasks resulting from the test runs are shown in table 5.2 for the different algorithms and test cases. Generally, the methods perform well, as compared to the goal of generating solutions having at least 95% of the tasks assigned. All the methods achieve this for all test cases except for case 3.

	AFR		AFS		TFS		TFR	
	med	max	med	max	med	max	med	max
Case 1	99.5	99.7	99.7	99.7	99.5	99.7	99.5	99.7
Case 2	99.5	99.6	99.2	99.2	99.4	99.5	99.2	99.5
Case 3	89.5	93.2	90.0	93.4	89.3	92.7	89.5	92.6
Case 4	97.7	97.9	97.7	97.7	97.7	97.7	97.7	97.9
Case 5	99.4	99.4	99.4	99.4	99.1	99.1	99.4	99.7
Case 6	98.9	99.5	99.0	99.0	98.9	99.6	98.8	99.5

Table 5.2: The percentage of assigned tasks for different test cases and methods. The best and median are presented. Underlined values are best in their category (med/max) per case.

The costs for the initial solutions generated by the different methods are presented in Table 5.3. For different test cases, different methods are best. If we look at the best found solutions (with minimum cost), the results show that TFR and AFR are best on two cases each while TFS and AFS are best on one each. On the other hand, AFS is best on four cases if we look at the median cost.

	AFR		AFS		TFS		TFR	
	med	min	med	min	med	min	med	min
Case 1	3 418	2 339	2 349	2 300	2 668	2 064	2 626	1 998
Case 2	3 800	2 850	5 500	5 500	4 350	3 200	5 775	3 750
Case 3	4 825	3 716	4 619	3 666	4 837	3 813	4 755	3 806
Case 4	3 375	3 160	3 364	3 357	3 407	3 394	3 412	3 215
Case 5	3 061	3 046	3 060	3 046	3 132	2 939	2 943	2 269
Case 6	2 338	1 438	2 275	2 182	2 345	1 189	2 401	1 374

Table 5.3: The median and minimum costs of the solutions received for the different test cases and algorithms

The average computation times are presented in Table 5.4. The time is separated between the appending methods (AFR, AFS, TFR, and TFS) and `FillGaps`. To form a complete solution both `FillGaps` and one of the appending methods have to be run and the total running time is the sum of the corresponding computation times. As can be seen, TFR is the fastest for all the cases, and for some cases TFS is equally fast. The randomizing variants (AFR, TFR) are also generally faster since shuffling is faster than sorting.

	AFR	AFS	TFS	TFR	`FillGaps`
Case 1	0.50	0.64	0.30	0.30	0.22
Case 2	2.98	4.70	1.14	0.54	2.34
Case 3	0.82	1.36	0.16	0.10	0.64
Case 4	0.04	0.04	0.02	0.02	0.02
Case 5	14.36	20.18	16.28	2.68	10.22
Case 6	5.70	7.30	2.78	2.44	3.18

Table 5.4: The average computation time in CPU seconds for the different algorithms and test cases.

To set the running times in perspective, test case 5 takes almost one hour for the method IMP, while test case 4 only takes a few seconds. Over all the methods take no more than 1–2% of the running time as compared to that of IMP.

The average running times increase with the number of tasks and aircraft. As described in Section 4.3, the time complexity for AFR, AFS and TFR is $O(|A||F|)$ and for TFS it is $O(|A||F| \log |A|)$. Even though TFS has a slightly higher theoretical complexity, in practice, it is still generally faster than AFR and AFS.

Since the running time increases with both number of aircraft and tasks, it is reasonable that test case 5 has a lot longer running times than the other test cases.

However, it is remarkable that TFR succeeds with that instance a lot faster than the other methods do.

The running times should only give a hint of the performance of the methods since there have been just a minimal effort to improve the running time by implementation details. However, there are a couple of reasons why TFR and TFS are faster than AFR and AFS. The main reason is that the loop over aircraft is exited whenever a feasible assignment is found. AFS also calculates the number of restrictions, which requires a call to the Rave model (explained in Section 3.1.2) for each task/aircraft pair.

In Figure 5.1, the cost results are presented as bar charts. The costs are divided into connection cost, cost of unassigned tasks and the penalty for violating cumulative constraints. The labels indicates the cost components of each test case. The connection cost varies very little between the methods but the cost of unassigned tasks varies more.

The quality of the lower bound is not very good for test cases 1, 5, and 6, since all other solutions have much higher connection cost than the lower bound.

For each case, the four first bars represent the cost components of the median solution, as well as the 10th and 90th percentile (the black interval). The percentile values show that there is sometimes a big variation of results even if the minimum and median cost are similar, particularly the AFR and AFS for case 4 have a much higher 90th percentile than both the median and 10th percentile.

In Figure 5.1, the results for the methods are compared with Jeppesen's method IMP, running IMP with initial solutions and the lower bound. The best found initial solution by each method is used as input to IMP. As can be seen, the results when using an initial solution is about the same or better as without an initial solution.

Recall, from Table 5.2, that test case 3 has a lower percentage of assigned flights than the other cases. Still, the method IMP is able to find a solution with about the same cost as the LB when running IMP with the 'bad' initial solutions (Figure 5.1, Case 3).

The aircraft in test case 3 and 4 have almost all the same number of restrictions. Therefore, the result is very similar for AFS and AFR for these cases.

5.2 Results for Cases With Hard Cumulative Constraints

The methods AFM and TFM (see Section 4.4) are evaluated on test cases 7 and 8 from Table 5.1.

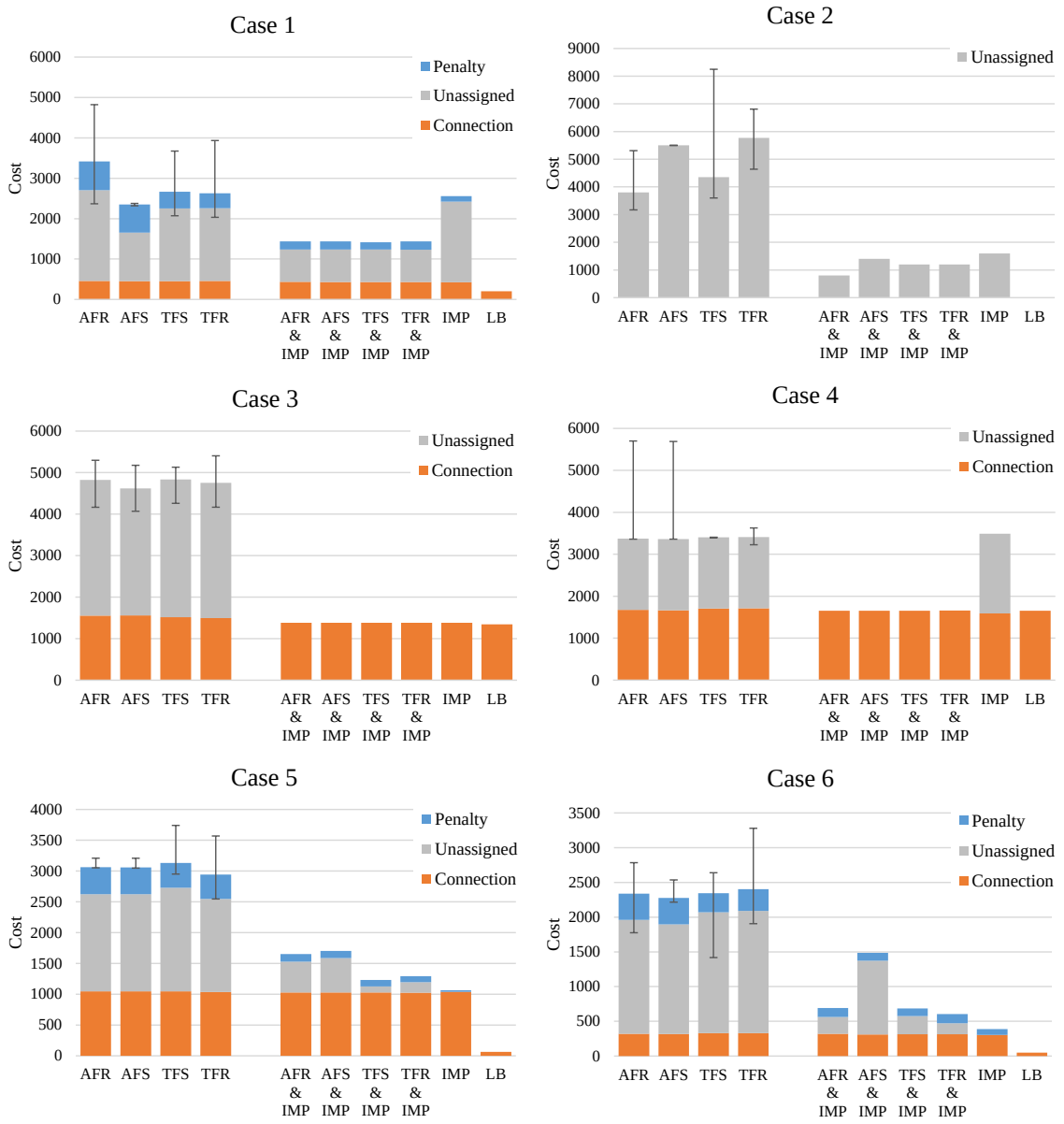


Figure 5.1: Bar plots of the result from the different greedy methods for six test cases, the solution with the median total cost is shown with the different cost components. The black interval represents the 10th and 90th percentiles, respectivly, of the total cost. The cost is also compared with Jeppesen's method IMP; see description in the text.

The naive method TFM shows a huge variation in solution quality. This is shown both in the percentages of assigned flights (Table 5.5) and the solution cost (Table 5.6).

Both TFM and AFM achieve the goal of assigned tasks for both test cases with respect to the best solution. For test case 8, AFM achieve the goal also with the median solution.

The running times are presented in Table 5.7. The results from the two methods

	TFM		AFM	
	med	best	med	best
Case 7	91.1	95.4	93.8	96.4
Case 8	58.4	97.9	98.3	98.8

Table 5.5: Percentage of assigned flights achieved for test cases 7 and 8. The values for the median and best solutions are presented.

	TFM		AFM	
	med	min	med	min
Case 7	2 170	1 496	1 764	1 335
Case 8	51 085	11 192	6 789	6 291

Table 5.6: Solution costs for test cases 7 and 8. The values for the median and best solutions are presented.

are similar for case 7 but differs more for case 8. For both cases, TFM is faster than AFM. However, TFM seems to take a lot longer to run for the same problem sizes than TFR. Therefore, it must be the evaluation of the cumulative constraints that accounts for most of the increase since this is the only difference between TFM and TFR.

	AFM	TFM	`FillGaps`
Case 7	12.80	11.18	0.40
Case 8	2.14	1.60	0.02

Table 5.7: The average computation time in CPU seconds for AFM and TFM for test cases 7 and 8.

In Figure 5.2 both the median and best results are shown and compared with the results achieved by the method IMP. The best solutions from AFM and TFM, respectivly, are used as input to IMP when run with an initial solution. Even though AFM finds a solution with lower cost than TFM for both test cases, it is when the solution produced by TFM is used in initial solution that the lowest cost after the optimization is achieved for case 8. However, the initial solutions from both methods fails to improve the final solution objective. Note that test case 7 has a penalty cost but it is too small to be visible in the figure.

Figure 5.2 shows that the 10th percentile for TFM is quite a bit higher than the objective value of the best solution for both test cases. This indicates that good solutions are rare and may not be found if the methods are only run a few times.

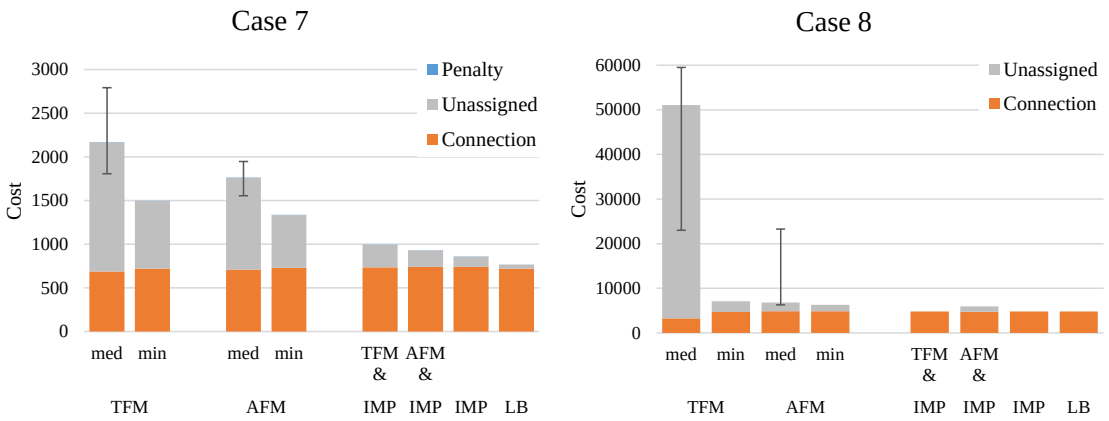


Figure 5.2: Bar plot with the different methods for test cases 7 and 8, the black interval represents the 10th and 90th percentiles, respectivly, of the total cost; see description in the text.

6

Discussion and Conclusion

This thesis presents some variations of fast heuristics to create initial solutions to the tail assignment problem. These solutions can be used to warm start an improvement method as well as to produce legal solutions fast for other use cases.

The developed methods consist of first applying the method called `FillGaps` that assigns tasks between pre-assignments and then applying one of the appending methods Aircraft First Random (AFR), Aircraft First Sort (AFS), Task First Sort (TFS), Task First Random (TFR), Task First Maintenance (TFM), and Aircraft First Maintenance (AFM). `FillGaps` and AFM use variants of heuristic version of Depth First Search (DFS) and the other methods use greedy algorithms.

The methods are chosen with short running time as the main criteria. Since the produced solutions are to be improved by an existing improvement method, there is no need to develop an own improvement algorithm. If some more complex technique than greedy algorithms and DFS was used, even more tasks could probably be assigned but it would be to the detriment of running time. This thesis focused on the fastest possible approaches.

The evaluation was performed on eight test cases, six without hard cumulative constraints and two with cumulative constraints. The methods performed well with respect to the percentage of tasks assigned. The goal was to assign 95% of the tasks and this was achieved with some method for all except one test case. On the other hand, the results of actually using the solutions as initial solutions to the improvement method (IMP) showed that it is not always better to use an initial solution. For the cases for which IMP produced a solution with all tasks assigned without an initial solution, using an initial solution may even produce a worse final solution. However, for the cases where IMP without initial solution did not assign all of the tasks, the results of using an initial solution improved the final solution. It should, however, be noted that more test cases are needed before this kind of conclusions can be drawn with certainty.

The objective value was not improved for the cases with hard cumulative constraints. Therefore, the methods for hard cumulative constraints need to be improved to be useful. However, AFM reached the goal for percentage of tasks assigned both for the minimum and median cost solution found for one test case and TFM reached the goal for both test cases with the minimum cost solution. Since the methods

achieve a high number of assigned tasks, it might be possible to tune Jeppesen's optimizer (using another improvement method than IMP) so that good results can be obtained with these methods as well.

For the improvement method to benefit from an initial solution, it might not be enough to have a high number of assigned tasks in the initial solution. Instead, there seems to be some other criteria. From the results presented in this thesis it is hard to draw any conclusions about why the final solution is not always improved by using an initial solution. But since the planning period is divided into time windows, it is possible that there is no way to cover all tasks in the interval with the fixed start and end positions that comes from the initial solution, while it may be possible to cover all tasks if the end position is not fixed. Therefore, it might be necessary to consider some other criteria while constructing the initial solutions.

The method IMP is just one out of many ways to run Jeppesen's optimizer. For example, IMP sweeps through the planning period only once, while it is more common to do several sweeps through the planning period. There might also be other configurations for the column generation part of the optimizer that can make better use of the initial solutions.

The solutions generated by the methods proposed in this thesis could also be used as initial solutions to other methods. For example, stochastic methods like Simulated Annealing and Particle Swarm Optimization, which Jamili et al. [9] use for other airline optimization problems, can benefit from the solutions produced by the methods proposed in this thesis. The solutions can also be used when designing the rules for a new airline customer to quickly get an idea of how the different rules and costs influence the solution.

Our methods are very fast, all of them are run in less than half a minute even for an instance with 112 aircraft. This indicates that it is possible to use our methods also for even larger instances. It also opens up for the possibility to run several different methods and to run each method several times.

Our results show that the orders of tasks and aircraft are important, and to get the best out of these methods, they should be run multiple times with different orderings. A possible improvement in the future is to find a sorting/ordering principle that works well for all cases or at least for a well defined set of problem instances. Some sorting schemes were not included in this thesis, since the preliminary results showed that they did not work very well. For methods based on the task-first principle, this includes sorting on arrival time, using the reverse order every second iteration, and choosing the task with the lowest connection cost.

The methods also need more testing in order to determine which method that performs best for which type of cases. There is also a need to test if there is a significant difference in the quality of the final solution of using the best or worst solution found by a method. Since the objective value of the final solution did not seem to correlate with the objective value of the initial solution, it might be the case that it is unnecessary to run the initial methods multiple times if the improvement method

ends up in a good solution also with a ”bad” initial solution.

Even though our evaluation has been focused on comparing the different appending algorithms, one should not forget the important part played by the method `FillGaps`. This method takes care of the pre-assigned tasks that the time window heuristic sometimes can have difficulties with.

The final conclusion is that the methods developed in this thesis create solutions with many assigned tasks fast, but more testing and analysis is needed to know how and when these initial solutions should be used.

Bibliography

[1] M. Grönkvist, "The Tail Assignment Problem", PhD thesis, Chalmers University of Technology, 2005.

[2] P. Belobaba, A. R. Odoni, and C. Barnhart, *The Global Airline Industry*, English, Second edition. Chichester, West Sussex, UK: John Wiley & Sons, 2015.

[3] S. Gabteni and M. Grönkvist, "Combining column generation and constraint programming to solve the tail assignment problem", *Annals of Operations Research*, vol. 171, no. 1, pp. 61–76, 2009. DOI: 10.1007/s10479-008-0379-1.

[4] O. Khaled, M. Minoux, V. Mousseau, S. Michel, and X. Ceugniet, "A compact optimization model for the tail assignment problem", *European Journal of Operational Research*, vol. 264, no. 2, pp. 548–557, 2018. DOI: 10.1016/j.ejor.2017.06.045.

[5] A. Sarac, R. Batta, and C. M. Rump, "A branch-and-price approach for operational aircraft maintenance routing", *European Journal of Operational Research*, vol. 175, no. 3, pp. 1850–1869, 2006. DOI: 10.1016/j.ejor.2004.10.033.

[6] N. Safaei and A. K. Jardine, "Aircraft routing with generalized maintenance constraints", *Omega*, 2017. DOI: 10.1016/J.OMEGA.2017.08.013.

[7] A. Kasirzadeh, M. Saddoune, and F. Soumis, "Airline crew scheduling: models, algorithms, and data sets", *EURO Journal on Transportation and Logistics*, vol. 6, no. 2, pp. 111–137, 2017. DOI: 10.1007/s13676-015-0080-x.

[8] S. Ruther, N. Boland, F. G. Engineer, and I. Evans, "Integrated Aircraft Routing, Crew Pairing, and Tail Assignment: Branch-and-Price with Many Pricing Problems", *Transportation Science*, vol. 51, no. 1, pp. 177–195, 2017. DOI: 10.1287/trsc.2015.0664.

[9] A. Jamili, "A robust mathematical model and heuristic algorithms for integrated aircraft routing and scheduling, with consideration of fleet assignment problem", *Journal of Air Transport Management*, vol. 58, pp. 21–30, 2017. DOI: 10.1016/J.JAIRTRAMAN.2016.08.008.

[10] G.-F. Deng and W.-T. Lin, "Ant colony optimization-based algorithm for airline crew scheduling problem", *Expert Systems with Applications*, vol. 38, no. 5, pp. 5787–5793, 2011. DOI: 10.1016/J.ESWA.2010.10.053.

[11] M. M. Wahde, *Biologically inspired optimization methods : an introduction*. WIT Press, 2008, p. 218.

[12] Y.-C. Lai, D.-C. Fan, and K.-L. Huang, "Optimizing rolling stock assignment and maintenance plan for passenger railway operations", *Computers & Industrial Engineering*, vol. 85, pp. 284–295, 2015.

[13] Z. A. Juman and M. A. Hoque, "An efficient heuristic to obtain a better initial feasible solution to the transportation problem", *Applied Soft Computing Journal*, vol. 34, pp. 813–826, 2015. DOI: 10.1016/j.asoc.2015.05.009.

[14] J. Joubert and S. Claasen, "A sequential insertion heuristic for the initial solution to a constrained vehicle routing problem", *ORiON: The Journal of ORSSA*, vol. 22, no. 1, pp. 105–116, 2006. DOI: 10.5784/22-1-36.

[15] P. C. Guedes and D. Borenstein, "Column generation based heuristic framework for the multiple-depot vehicle type scheduling problem", *Computers and Industrial Engineering*, vol. 90, pp. 361–370, 2015. DOI: 10.1016/j.cie.2015.10.004.

[16] W. Dai, X. Sun, and S. Wandelt, "Finding feasible solutions for multi-commodity flow problems", in *Chinese Control Conference, CCC*, vol. 2016-Augus, IEEE, 2016, pp. 2878–2883. DOI: 10.1109/ChiCC.2016.7553801.

[17] S. Boyd and L. Vandenberghe, *Convex Optimization*. Cambridge, United Kingdom: Cambridge University Press, 2004.

[18] T. C. Hu and A. B. Kahng, "Introduction to the Simplex Method", in *Linear and Integer Programming Made Easy*, Cham: Springer International Publishing, 2016, pp. 39–60. DOI: 10.1007/978-3-319-24001-5_4.

[19] D. A. Spielman and S.-H. Teng, "Smoothed analysis of algorithms: Why the simplex algorithm usually takes polynomial time", *J. ACM*, vol. 51, no. 3, pp. 385–463, 2004. DOI: 10.1145/990308.990310.

[20] R. Vershynin, "Beyond Hirsch conjecture: Walks on random polytopes and smoothed complexity of the simplex method", English, *SIAM Journal on Computing*, vol. 39, no. 2, pp. 646–678, 2009.

[21] J. Matoušek and B. Gärtner, "Not only the simplex method", in *Understanding and Using Linear Programming*, Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, pp. 105–130. DOI: 10.1007/978-3-540-30717-4_7.

[22] ——, "Integer programming and LP relaxation", in *Understanding and Using Linear Programming*, Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, pp. 29–40. DOI: 10.1007/978-3-540-30717-4.

[23] G. Nemhauser and L. Wolsey, "Integer programming", in *Handbooks in Operations Research and Management Science: Optimization*, G. Nemhauser, A. Rinnooy Kan, and M. Todd, Eds. Amsterdam, The Netherlands: Elsevier Science Publishers, 1989.

[24] J. Desrosiers and M. E. Lübbecke, "A primer in column generation", in *Column Generation*, G. Desaulniers, J. Desrosiers, and M. M. Solomon, Eds. Boston, MA: Springer US, 2005, pp. 1–32. DOI: 10.1007/0-387-25486-2_1.

[25] S. Irnich and G. Desaulniers, "Shortest path problems with resource constraints", in *Column Generation*, G. Desaulniers, J. Desrosiers, and M. M. Solomon, Eds. Boston, MA: Springer US, 2005, pp. 33–65. DOI: 10.1007/0-387-25486-2_2.

[26] M. E. Lübbecke and J. Desrosiers, "Selected topics in column generation", *Operations Research*, vol. 53, no. 6, pp. 1007–1023, 2005. DOI: 10.2307/25146936.

[27] "Artificial variables", in *Encyclopedia of Operations Research and Management Science*, S. I. Gass and M. C. Fu, Eds., Boston, MA: Springer US, 2013, pp. 83–83. DOI: 10.1007/978-1-4419-1153-7_200964.

[28] C. Barnhart, E. L. Johnson, G. L. Nemhauser, M. W. P. Savelsbergh, and P. H. Vance, "Branch-and-price: Column generation for solving huge integer programs", *Operations Research*, vol. 46, no. 3, pp. 316–329, 1998. DOI: 10.1287/opre.46.3.316.

[29] "Greedy algorithm", in *Encyclopedia of Operations Research and Management Science*, S. I. Gass and M. C. Fu, Eds., Boston, MA: Springer US, 2013, pp. 666–667. DOI: 10.1007/978-1-4419-1153-7_200276.

[30] G. Brassard and P. Bratley, "Greedy algorithms", in *Fundamentals of algorithmics*. Englewood Cliffs: Prentice Hall, 1996, vol. 33.

[31] E. Andersson, A. Forsman, S. E. Karisch, N. Kohl, and A Sørensson, "Problem solving in airline operations", *Carmen Research and Technology Report CRTR-0404, Carmen Systems AB, Gothenburg, Sweden*, 2004.

[32] M. Grönkvist, "Accelerating column generation for aircraft scheduling using constraint propagation", *Computers & Operations Research*, vol. 33, no. 10, pp. 2918–2934, 2006. DOI: 10.1016/J.COR.2005.01.017.

[33] J. Kleinberg and E. Tardos, *Algorithm Design*, English, Pearson New International Edition. Harlow, United Kingdom: Pearson Education M.U.A, 2013.