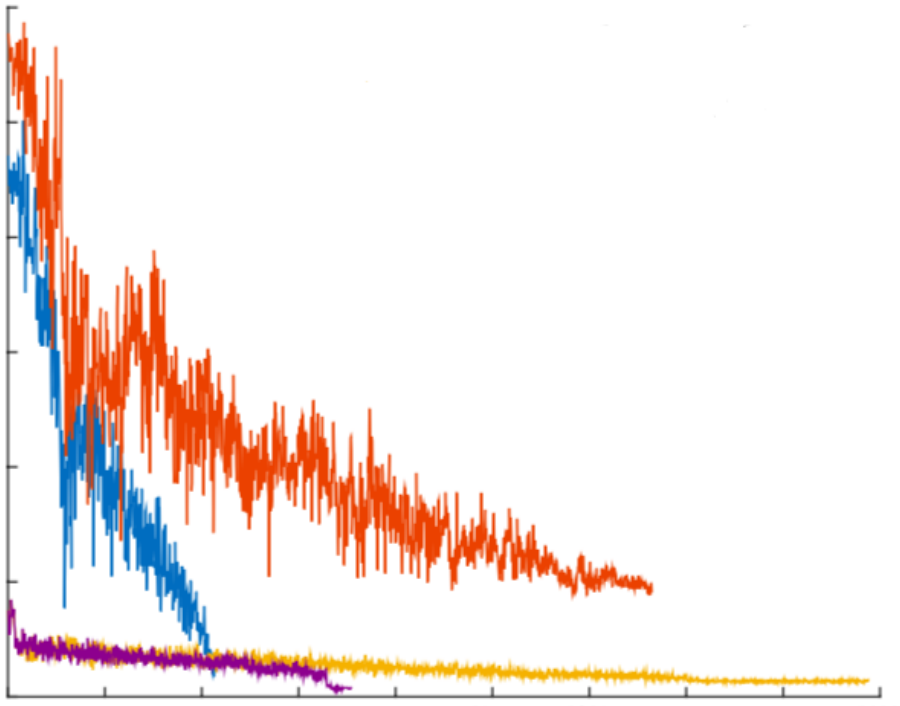


The tail assignment problem for single and mixed aircraft fleets

Mathematical modeling, solution, and implementation

Master's thesis in Engineering Mathematics

Mattias Danielsson, Gustav Karlsson

Department of Mathematical Sciences
CHALMERS UNIVERSITY OF TECHNOLOGY AND UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2018

Thesis for the Degree of Master of Science

The tail assignment problem for single and mixed aircraft fleets
Mathematical modeling, solution, and implementation

Mattias Danielsson, Gustav Karlsson

UNIVERSITY OF
GOTHENBURG

CHALMERS

Department of Mathematical Sciences
Division of Applied Mathematics and Statistics
Chalmers University of Technology and University of Gothenburg
Gothenburg, Sweden 2018

The tail assignment problem for single and mixed aircraft fleets
Mathematical modeling, solution, and implementation
Mattias Danielsson, Gustav Karlsson

Supervisor: Ann-Brith Strömberg, Mathematical Sciences
Examiner: Ann-Brith Strömberg, Mathematical Sciences

Thesis for the Degree of Master of Science
Department of Mathematical Sciences
Division of Mathematics
Chalmers University of Technology and University of Gothenburg
SE-412 96 Gothenburg

Tail assignment problem for single/mixed aircraft fleets: modeling, solution and implementation
Mattias Danielsson, Gustav Karlsson
Department of Mathematical Sciences
Chalmers University of Technology and University of Gothenburg

Abstract

The tail assignment problem is the problem to, in accordance with some optimization criterion, assign a set of flights to a set of aircraft in order to create a feasible flight schedule for an airline. In this thesis we approach this problem using column generation combined with heuristic methods to enforce an integer solution. Apart from the more well established column generation method derived from Dantzig–Wolfe decomposition, a new column generation is used, the recently developed *integer quality column generation* [6] (Bredström, Jörnsten, Rönnqvist, Bouchard: Int. Trans: Oper. Res. 21, 177-197, 2014). To enforce integer solutions, we employ a connection fixing heuristic. Enabled by our proposed modeling of the flight network, in which the initial positions of the aircraft are represented by nodes, we also develop and propose an aircraft fixing heuristic. We have, further, developed new pre-processing methods for reducing the number of arcs in the network (and thereby reducing the solution space for the subproblem). Our results show that the standard column generation together with either of the integer heuristics are generally preferable, except for some cases where the integer quality column generation produced better results. Our new pre-processing methods show promising results, for some test instances significantly reducing the number of arcs in the network.

Acknowledgements

We would like to thank our supervisor and examiner Ann-Brith Strömberg for guidance through the work on this thesis. At Aviolinx, we would like to thank Joakim Andersson and Per Genell for insight into the airline business, and Lukasz Rosikiewicz for helping us with anything regarding C#.

Contents

1
Introduction

1.1 Background

This thesis concerns the development and implementation of mathematical optimization models to solve aircraft scheduling problems with several different aircraft types. The project has been done in cooperation with Aviolinx. Aviolinx is a middle-sized company based in Stockholm that offers a software service tool for administration of operation of small and middle sized airlines.

The aircraft scheduling problem considered is the so called *tail assignment problem*[1] [13], which is to, given a set of flights and a set of aircraft, assign one aircraft to each flight according to some optimization criterion. The aircraft routes should fulfill some appropriate feasibility requirements. For example, the solution must consist of continuous routes for each aircraft, and there must be enough ground time between the flights along each route, such that passengers can be loaded/unloaded and the plane can be refueled. The feasibility requirements also includes that each aircraft undergoes maintenance at specific locations and during specific time intervals, to satisfy international security regulations.

There are several reasons why an airline wants to optimize its schedules, the most obvious perhaps being to reduce costs, but the proposed algorithm can also be used, e.g., to make an aircraft schedule better match that of the crew, or to reduce the time that the aircraft spend on the ground.

Aviolinx have provided us with data, on which the models have been tested, as well as the overall interface, RAIDO [3], in which our algorithm has been implemented. The algorithm is implemented in C#.

1.2 Literature study

A thorough look at the tail assignment problem is presented in the PhD thesis [13] by Grönkvist from 2005, which introduces the problem and provides a mathematical programming as well as a constraint programming approach to solve it. Among the things discussed in the mathematical programming part of the thesis is a column generation algorithm, solution methods for the associated subproblem, and heuristic methods to obtain integer solutions. The work presented in this report is to a large

[1] For airlines, the terms aircraft, vehicle, and tail are used interchangeably (tail referring to the unique tail number associated with each aircraft). Through this report we use the word aircraft, except when referring to the *tail assignment problem*, which is the established name for the considered optimization problem.

extent built on [13], and extends on some parts of it. A combined column generation and constraint programming approach is presented by Gabteni and Grönkvist ([10]); when applying integer heuristics to the mathematical programming model, in order to avoid unassigned flights it adds feasibility checks within a constraint programming model.

The book Column Generation ([7]) by Desaulniers, Desrosiers and Solomon addresses column generation as a solution method for many different large-scale optimization problems. It includes a chapter on models in the airline industry, where an outline for solving aircraft routing problems is presented. Among other things, the label setting algorithm used to solve the subproblem which arises in the tail assignment problem is described.

Froyland, Maher and Wu have developed a method to solve *The Recoverable Robust Tail Assignment Problem* ([9]). It is often the case that delays will spread if one flight is late. For example, the crew on a delayed flight are likely to arrive late to the next flight they are to serve on. The aim of solving the recoverable robust tail assignment problem is to generate aircraft schedules in which these kinds of events are less likely to happen, and in the case of disruptions, the schedule should be recoverable, such that re-planning can be done at a low additional cost.

A model for optimizing a schedule to minimize fuel costs has been developed by Lapp and Wikenhauser ([15]), in which they associate with each aircraft a score which depends on the fuel efficiency of the engines. This score can be used to optimize towards an aircraft schedule in which the more efficient aircraft are used more frequently than their less efficient counterparts. They argue that the fuel costs constitute a large part of an airline's expenditure, and that great savings can be made by optimizing the aircraft schedule in this manner. Lapp and Wikenhauser use a model which is similar to the one presented in [13], but also a simpler model, in which complete routes have already been established, and the problem consists of optimally assigning one aircraft to each of these routes.

Borndörfer, Dovica, Nowak and Schickinger in [5] present a stochastic optimization method for solving the tail assignment problem with the objective to minimize delays. They model *primary delays* (which are due to, for example, bad weather, for which there is no avoidance) as random variables, from which random variables for secondary, propagated delays are derived. The aim is to create routes for aircraft with a minimum probability for propagated delays. A thesis on the same subject has been written by Dovica in 2014 ([8]).

In the article ([19]) from February 2017, Ruther, Boland, Engineer and Evans present an integrated aircraft routing, crew pairing and tail assignment problem, as well as a method to solve it. They solve this integrated problem by simultaneously generating routes for aircraft and pairings for crew members, and apply penalties to solutions in which the crew have a short connection time between two flights being assigned to two different aircraft. By minimizing the number of such short crew connections, the aim is to create a robust schedule in which possible disruptions create less of a hassle.

Some new methods for solving set partitioning problems (as which we will formulate the tail assignment problem) have been developed in later years. These include the works by Rönnberg and Larsson ([20]), and Bredström, Jörnsten, Rönnqvist,

and Bouchard ([6]). We will go into more detail about the latter of these methods in Section 3.3.

1.3 Project scope

We start by formulating the general minimum cost multi-commodity network flow problem and the tail assignment problem as a resource constrained multi-commodity network flow problem.

Thereafter we present the methods needed to solve the tail assignment problem and the necessary theoretical background needed in order to apply the methods to the problem.

The main solution technique used is that of column generation. To be able to apply this technique, we decompose the tail assignment problem into a master problem and a subproblem using Dantzig-Wolfe decomposition. The subproblem is a resource constrained shortest path problem, which we solve using dynamic programming. The master problem is a continuously relaxed set partitioning problem, which is solved using a commercial solver.

Two different column generation methods are used, the standard column generation and the IQ column generation.

To get an integer solution to the master problem, we use heuristic methods, which after specified conditions are fulfilled, restricts the solution space to the subproblem and the master problem.

We present methods to remove unnecessary arcs in graph representing the subproblem. These methods can be applied to the network of nodes before running a column generation algorithm, and are (under some basic assumptions) guaranteed not to remove the optimal solution from the solution space.

We also present a simplified model, which can be used to compute a lower bound to the tail assignment problem, and to roughly monitor the convergence of the column generation.

Our solution methods and algorithms are evaluated on data from commercial airlines.

1.4 Limitations

To expect reasonable results from the pre-processing methods presented in Section 4.4, it is required that a solution exists in which all flights during the considered time period are covered by aircraft. We can rule out the possibility that such a feasible solution exists, if the solution to the minimum cost network flow problem described in Section 4.3 contains unassigned flights. We will, however, not study any additional methods for checking problem feasibility. It should be noted that the column generation algorithms can be run even if no solution exists with all flights assigned to aircraft. In these cases the algorithm will inevitably minimize the number of unassigned flights in the final aircraft schedule (due to the high costs put on unassigned flights).

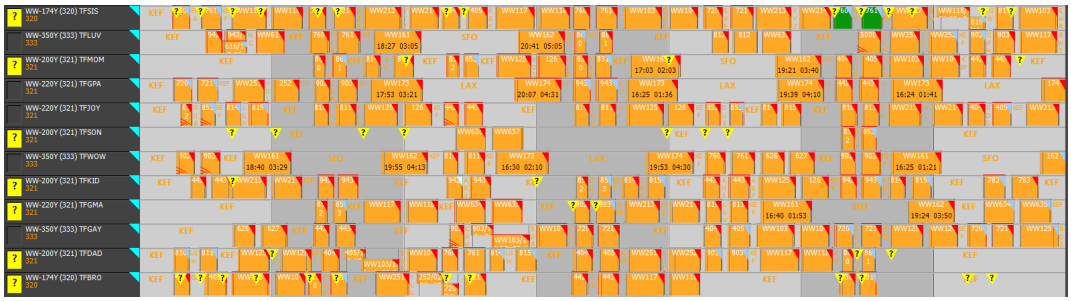


Figure 1.1: An example of a fleet schedule over the course of a week. Each row represents an aircraft. Orange and green boxes represent flights assigned to the aircraft represented by the row on which they appear. A box placement on the horizontal axis specifies at what time the flight depart and land.

For the work presented in this thesis to be used in the airline industry, a user interface has to be constructed. What explicitly should be put in such an interface is not discussed in this thesis.

Parts of the intended methods of solution will consist of heuristics, and as a result optimal solutions can not be guaranteed. Methods which can be used to obtain the optimal solution to the tail assignment problem, such as branch-and-price, are only superficially discussed within this thesis.

Figure 1.1 shows an example of a one week aircraft schedule for a smaller airline as a gantt-chart. The time span over which the flight scheduling should be done can vary from a single day to one year. As the tail assignment problem quickly grows very large when the number of flights is increasing, our project task is limited to scheduling over a time span of maximum a couple of months, depending on the fleet size.

2

Network flow formulation of the tail assignment problem

In Section 2.2 we formulate the general minimum cost multi-commodity network flow problem, and in Section 2.3 we formulate the tail assignment problem as a resource constrained multi-commodity network flow problem.

2.1 Problem formulation

The tail assignment problem is the problem of, given a set of flights and a set of aircraft, assign an aircraft to each flight, and thus create a flight schedule for an airline. When creating the schedule one has to take several things into account. The current status of the aircraft is one such thing. There are several different maintenance requirements which needs to be fulfilled for each aircraft if the airline wants to use it commercially. All aircraft need to go through different maintenance checks after every n-th flying hour and after every m-th landing and take off. The aircraft can be of such nature that it cannot take off from a specific airport due to, for example, the length of its runway. Furthermore, a smaller ground check needs to be done between each flight and the aircraft have to be refueled. These checks are called turn arounds and the time needed to perform a turn around is generally less than an hour. In order to create a feasible schedule one has to make sure that all of these requirements are fulfilled.

Even if a schedule fulfills all of these requirements it is not necessarily true that it is appealing to the airline. The airline wants to have a schedule that is as money efficient as possible. The routes for each aircraft should be continuous, in that when an aircraft lands on a specific airport, its next flight should depart from that same airport, as it would be too expensive to transport empty aircraft between airports. Another way for them to save money is to reduce the time spent on ground of a utilized aircraft. It is also appealing for the flight crew to have a lean schedule without long stays at each airport.

This motivated an objective function that gets more expensive the more ground time a flight path has. We have chosen an objective function that is a linearly increasing function of the number of minutes the aircraft stays on ground, with its lowest value for the minimum required turn around time between two flights.

Ideally, every flight in the time period should be assigned to an aircraft. To enforce this, we put high costs on solutions in which flights are left unassigned.

2.2 The minimum-cost multicommodity network flow problem

The minimum-cost multicommodity network flow problem ([21]) is a generalization of the minimum-cost network flow problem, where we have several "commodities" which should flow through the network. In the model (2.1) we denote the set of nodes by I, the set of directed arcs by $A \subseteq I \times I$, the set of commodities by T, the source node for commodity t as $i_0^t \in I$, the sink node for commodity t as $i_N^t \in I$, the cost of sending one unit of flow of commodity t on arc (i,j) as c_{ijt}, the flow capacity of arc (i,j) as k_{ij}, and the amount of commodity t sent on arc (i,j) as x_{ijt}:

$$\underset{x_{ijt}}{\text{minimize}} \quad z = \sum_{i \in I} \sum_{j \in I} \sum_{t \in T} c_{ijt} x_{ijt}, \tag{2.1a}$$

$$\text{s.t.} \quad \sum_{j \in I} x_{jit} - \sum_{j \in I} x_{ijt} = \begin{cases} -q_t, & i = i_0^t, \\ q_t, & i = i_N^t, \\ 0, & i \in I \setminus \{i_0^t, i_N^t\}, \end{cases} \quad i \in I, t \in T, \tag{2.1b}$$

$$\sum_{t \in T} x_{ijt} \leq k_{ij}, \quad i \in I, j \in I \tag{2.1c}$$

$$x_{ijt} \geq 0, \quad i \in I, j \in I, t \in T. \tag{2.1d}$$

The constraints (2.1b) state that we should have q_t units of flow leaving the source node and entering the sink node for commodity t. It further states that we should have flow balance for every other node in the network. Constraints (2.1c) state that the flow on each arc should not exceed the corresponding arc capacity.

2.3 The tail assignment problem modelled as a resource constrained multicommodity network flow problem

With aircraft seen as commodities, the tail assignment problem can be modelled as a resource constrained multicommodity network flow problem (see [4]). In such a model, each node corresponds to either a specific flight or a specific aircraft, the (directed) arcs represent possible connections between nodes (see Figure 2.1) and each path in the network corresponds to a set of flights assigned to an aircraft. By "path", we mean a set of n connected nodes in the network $\{i_{k_1}, i_{k_2}, ..., i_{k_{n-1}}, i_{k_n}\}$. A complete path must include the start node i_0^t as the first node and the sink node i_N^t as the last node.

We let I be the set of nodes, where i_0 is the source node, and i_N the sink node in the network. Note that in the general multicommodity network flow formulation there is one source node and one sink node per commodity, while in our formulation the commodities share the same source and sink node. Instead, every commodity (aircraft) has one aircraft node which each requires one unit of flow from the source

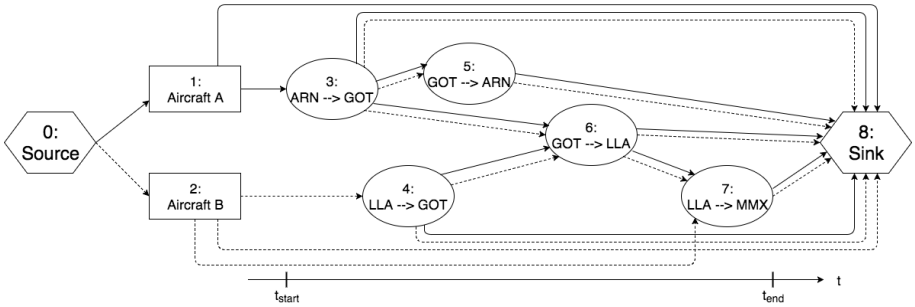


Figure 2.1: A small network representing two aircraft (nodes 1 and 2) and five flights (nodes 3 through 7) departing in the time between t_{start} and t_{end} with their horizontal placement suggesting different operation times. Arcs which has a flow capacity of 1 are displayed in this figure. Arcs related to aircraft A are depicted as filled lines, while arcs related to aircraft B are depicted as dashed lines. One feasible solution to the multi-commodity network flow problem (2.5) for this network are the paths defined by the ordered set of nodes $\{0, 1, 3, 5, 8\}$ and $\{0, 2, 4, 6, 7, 8\}$. The arcs related to aircraft B originating from flight 3 and 5 can not be used in any solution, but will be present in the model if they are not found and removed using a pre-processing method.

node. The source node only has arcs to the aircraft nodes. The aircraft nodes are elements of the set $I_{\text{a}} \subset I$, and $I_f := I \backslash \{I_{\text{a}}, i_0, i_N\}$ is the set of flight nodes. The set of available aircraft is denoted as T. Note that $|I_{\text{a}}| = |T|$. The flow capacity on arc (i, j) for aircraft t is denoted by e_{ijt}, which equals 1 if aircraft t can take flight j directly after flight i, and zero otherwise.

The decision variable x_{ijt} equals one if there is a positive flow on the edge between node i and node j, and passing through aircraft node t, and zero otherwise[1]. As an example, looking at Figure 2.1, x_{361} would equal 1 if flight 6 directly succeeds flight 3 in a path taken by aircraft 1. The decision variables are formally defined as

$$x_{ijt} = \begin{cases} 1, & \text{if flight } j \text{ is taken directly after flight } i \text{ by aircraft } t, \\ 0, & \text{otherwise,} \end{cases} \tag{2.2}$$

for $i \in I, j \in I, t \in T$.

In the following, we will use m to denote a type of resource which can be consumed. The consumption accumulates when adding flights to a path, and for which, before a limit has been reached, maintenance of a specific sort has to be performed. For example, if maintenance has to be performed at least after every 100 flying hours, *the total hours flown since last maintenance check* would be considered a resource, with a limit of 100. Due to the one-to-one correlation between resources and their corresponding maintenance activities, we will use m to denote both of these.

Let M denote the set of maintenance activities. β_{mt} is the upper bound on how much of resource m can be consumed by aircraft t before maintenance of type m must be done. We let the variables s_{jmt} denote the consumption of resource m for

[1] Aircraft node t corresponds to the source node for commodity t in the multi-commodity network flow problem (2.1).

aircraft t caused by taking flight j. Given a path in the network leading up to flight j, r_{jmt} denotes the total amount of resource m consumed by aircraft t on the path leading up to, and including flight j (where $x_{ijt} = 1$ for some preceding flight i), and is iteratively defined as

$$r_{jmt} = \begin{cases} s_{0mt}, & \text{if } j \text{ is an aircraft node,} \\ s_{jmt}, & \text{if maintenance of type } m \text{ can be done on aircraft } t \text{ between} \\ & \text{flight node } j \text{ and its predecessor } i, \\ r_{imt} + s_{jmt}, & \text{if maintenance of type } m \text{ could not be done} \\ & \text{on aircraft } t \text{ between flight node } j \text{ and its predecessor } i, \end{cases} \tag{2.3}$$

for $i \in I, j \in I, t \in T$. s_{0mt} is the amount of resource m already consumed by aircraft t at the start of the time span of which the tail assignment should be made. To express the resource consumption in terms of the decision variables x_{ijt} we introduce constants v_{ijmt} which equals to one if maintenance m can be done between flights i and j on aircraft t and zero otherwise. The accumulated consumption of resource m up to flight j by aircraft t can then be expressed as

$$r_{jmt} = s_{jmt} + \sum_{i \in I}(1 - v_{ijmt}) r_{imt} x_{ijt}, i \in I, j \in I, t \in T, \tag{2.4}$$

meaning that the resource consumption of resource m on node j by aircraft t is the resource consumed on flight j, s_{jmt}, plus the accumulated resource consumption on node i, if flight j is directly succeeding flight i and maintenance of type m is not done between the two flights.

We note that the resource constraints as stated here are non-linear. In section 2.3.1, we will show that there is an equivalent linear formulation of these constraints.

The parameter c_{ijt} denotes the cost of assigning flight j directly after flight i to aircraft t. The mathematical formulation of the tail assignment problem reads as follows:

$$\begin{aligned}
\underset{x_{ijt}}{\text{minimize}} \quad & z = \sum_{i \in I}\sum_{j \in I}\sum_{t \in T} c_{ijt} x_{ijt}, && && \text{(2.5a)} \\
\text{s.t.} \quad & \sum_{j \in I} x_{jit} - \sum_{j \in I} x_{ijt} = 0, && t \in T, i \in I \backslash \{i_0, i_N\}, && \text{(2.5b)} \\
& \sum_{t \in T}\sum_{j \in I} x_{ijt} = 1 && i \in I \backslash \{i_0, i_N\}, && \text{(2.5c)} \\
& x_{ijt} \leq e_{ijt}, && i \in I, j \in I, t \in T, && \text{(2.5d)} \\
& r_{jmt} = s_{jmt} + \sum_{i \in I}(1 - v_{ijmt}) r_{imt} x_{ijt}, && j \in I, t \in T, m \in M, && \text{(2.5e)} \\
& r_{imt} \leq \beta_{mt}, && i \in I, m \in M, t \in T, && \text{(2.5f)} \\
& x_{ijt} \in \{0, 1\}, && i, j \in I, t \in T, && \text{(2.5g)}
\end{aligned}$$

The objective function (2.5a) of the network flow problem is that of minimizing the total cost of the aircraft routes. The constraints (2.5b) demands that the aircraft

routes are continuous, in that if a flight arriving to a certain airport is assigned to exactly one aircraft, the next (if any) flight assigned to that aircraft must depart from that same airport. The constraints (2.5c) demands that every flight is assigned to an aircraft. These constraints correspond to the constraints (2.1c), and the constraints regarding the source and sink nodes in (2.1b), in the ordinary multicommodity network flow model. The constraints (2.5d) demands that there can only be flow on the arc (i, j) for aircraft t if aircraft t can take flight j directly after flight i. The cumulative constraint (2.5f) makes sure that no aircraft violates any maintenance restrictions. We note that this model leaves open the possibility to leave flight i unassigned with cost c_{iit}, if one adds an arc (i, i).

Note that, for a given $j \in I$, setting $e_{ijt} = 0$ for all $t \in T \setminus \{\tilde{t}\}$, captures the possibility to have flight j preassigned to aircraft $\tilde{t}$.

2.3.1 A linear formulation of the tail assignment problem

To show that there is a linear formulation of the resource constraints (2.5f), we introduce path variables in the model. We let P be the set of all complete paths in the network, starting at the source node, and ending at the sink node. Further, we let $\tilde{P} \subset P$ be the set of paths which violate a resource constraint, ($r_{imt} > \beta_{mt}$ for some node i in the path). We define constants

$$\delta_{ijtp} = \begin{cases} 1 & \text{if arc } (i,j) \text{ is used in path } p \text{ for aircraft } t \\ 0 & \text{otherwise,} \end{cases} \tag{2.6}$$

and binary decision variables

$$y_p = \begin{cases} 1 & \text{if path } p \text{ is used in the solution,} \\ 0 & \text{otherwise.} \end{cases} \tag{2.7}$$

A linear formulation of the tail assignment problem can then be formulated as to

$$\begin{aligned}
\underset{x_{ijt}}{\text{minimize}} \quad & z = \sum_{i\in I}\sum_{j\in I}\sum_{t\in T} c_{ijt}x_{ijt}, && && \text{(2.8a)}\\
\text{s.t.} \quad & \sum_{j\in I} x_{jit} - \sum_{j\in I} x_{ijt} = 0, && t \in T, i \in I\setminus\{i_0, i_N\}, && \text{(2.8b)}\\
& \sum_{t\in T}\sum_{j\in I} x_{ijt} = 1, && i \in I\setminus\{i_0, i_N\}, && \text{(2.8c)}\\
& x_{ijt} \le e_{ijt}, && i \in I, j \in I, t \in T, && \text{(2.8d)}\\
& y_p = 0, && p \in \tilde{P}, && \text{(2.8e)}\\
& x_{ijt} - \sum_{p\in P} \delta_{ijtp} y_p = 0, && i \in I, j \in I, t \in T, && \text{(2.8f)}\\
& y_p \in \{0,1\}, && p \in P, && \text{(2.8g)}\\
& x_{ijt} \in \{0,1\}, && i, j \in I, t \in T. && \text{(2.8h)}
\end{aligned}$$

The new constraint (2.8f) demands that, for every arc variable $x_{ijt} = 1$, exactly one of the path variables which include arc (i, j) for aircraft t is set to one. If

$x_{ijt} = 0$, all path variables which include arc (i, j) for aircraft t must be zero. Constraint (2.8e) ensures that a path variable is set to zero if its corresponding path violates a resource constraint. Since the tail assignment problem stated as (2.5) has an equivalent linear formulation in (2.8), Dantzig–Wolfe decomposition ([7]) and column generation can be applied.

3

Dantzig–Wolfe decomposition and column generation

We describe the theory of Dantzig-Wolfe decomposition for linear programs in Section 3.1. In Section 3.2, we relax the integer constraints on our resource constrained multi-commodity network flow problem and apply Dantzig-Wolfe decomposition to the relaxed problem. Applying this decomposition to our problem yields a set partitioning problem as the master problem, and resource constrained shortest path problems as the subproblems.

The main method we will use to solve the tail assignment problem is called column generation. We derive the column generation method in the Dantzig-Wolfe decomposition section and the integer quality column generation is further explained in Section 3.3. Our contribution is partly that of generalizing the modeling and solution approach from [6] to the tail assignment problem and compare it with a more conventional approach to column generation.

3.1 Dantzig-Wolfe decomposition

The Dantzig-Wolfe decomposition is a method of solving large-scale linear optimization problems by formulating them in a way that allows for a decomposition into smaller/simpler problems which can be solved iteratively.

Suppose that we have the optimization problem to

$$\begin{aligned} \underset{\mathbf{x}}{\text{minimize}}\ z := \quad & \mathbf{c}^{\mathrm{T}}\mathbf{x}, && \text{(3.1a)}\\ \text{s.t} \quad & \mathbf{A}\mathbf{x} \leq \mathbf{b}_1, && \text{(3.1b)}\\ & \mathbf{B}\mathbf{x} \leq \mathbf{b}_2, && \text{(3.1c)}\\ & \mathbf{x} \geq \mathbf{0}^n, && \text{(3.1d)} \end{aligned}$$

with $\mathbf{c} \in \mathbb{R}^n$, $\mathbf{x} \in \mathbb{R}^n$, $\mathbf{A} \in \mathbb{R}^{m_1 \times n}$, $\mathbf{B} \in \mathbb{R}^{m_2 \times n}$, $\mathbf{b}_1 \in \mathbb{R}^{m_1}$, and $\mathbf{b}_2 \in \mathbb{R}^{m_2}$. Let X be the polyhedron defined by constraints (3.1b) and (3.1d):

$$X = \{\mathbf{x} : \mathbf{A}\mathbf{x} \leq \mathbf{b_1}, \mathbf{x} \geq \mathbf{0}\}. \tag{3.2}$$

We assume that X is nonempty and bounded. Further, suppose that the constraints (3.1b) are of such a nature that the optimization problem defined by (3.1a), (3.1b), and (3.1d) could be solved relatively fast, while (3.1c) makes the problem far more complex to solve.

The set X has a finite number of extreme points, $\mathbf{x}^{(i)}, i = 1, ..., N$, where N is large. Every point $\mathbf{x} \in X$ can be expressed as a convex combination of the extreme points as, $\mathbf{x} = \sum_{i=1}^{N} \lambda_i \mathbf{x}^{(i)}$, where $\sum_{i=1}^{N} \lambda_i = 1$ and $\lambda_i \geq 0,\ i = 1, \ldots, N$. The optimization problem (3.1) can then be equivalently expressed as

$$\underset{x_i}{\text{minimize}}\ z_{\text{MP}} := \quad \mathbf{c}^{\text{T}} \sum_{i=1}^{N} \lambda_i \mathbf{x}^{(i)}, \tag{3.3a}$$

$$\text{s.t} \quad \sum_{i=1}^{N} \lambda_i \mathbf{B}\mathbf{x}^{(i)} \leq \mathbf{b_2}, \tag{3.3b}$$

$$\sum_{i=1}^{N} \lambda_i = 1, \tag{3.3c}$$

$$\lambda_i \geq 0, \quad i = 1, \ldots, N, \tag{3.3d}$$

The problem (3.3), is called the *Master Problem* (MP). This formulation enables us to define the *Restricted Master Problem* (RMP) where we only consider a subset of the extreme points in X. This is of interest because the set of extreme points in X is very large and impractical to deal with as a whole, sometimes even impossible. Given a feasible solution $\hat{\mathbf{x}}$ to (3.3), there exists a set of extreme points, $\mathbf{x}^{(i)}, i = 1, ..., m$, and corresponding λ_i fulfilling constraints (3.3c) and (3.3d), such that $\hat{\mathbf{x}}$ can be explicitly expressed as $\hat{\mathbf{x}} = \sum_{i=1}^{m} \lambda_i \mathbf{x}^{(i)}$, where $m \leq N$. Defining the set $\bar{X}$ as the convex hull of these extreme points,

$$\bar{X} = \Big\{\bar{\mathbf{x}} : \bar{\mathbf{x}} = \sum_{i=1}^{m} \lambda_i \mathbf{x}^{(i)}, \lambda_i \geq 0, \sum_{i=1}^{m} \lambda_i = 1\Big\}, \tag{3.4}$$

one can formulate the *Restricted Master Problem*, using $\lambda_i, i = 1, \ldots, m$, as variables, as to

$$\underset{\lambda_i}{\text{minimize}}\ z_{\text{RMP}} := \quad \mathbf{c}^{T} \sum_{i=1}^{m} \lambda_i \mathbf{x}^{(i)}, \tag{3.5a}$$

$$\text{s.t} \quad \sum_{i=1}^{m} \lambda_i \mathbf{B}\mathbf{x}^{(i)} \leq \mathbf{b_2}, \tag{3.5b}$$

$$\sum_{i=1}^{m} \lambda_i = 1, \tag{3.5c}$$

$$\lambda_i \geq 0, \quad i = 1, \ldots, m, \tag{3.5d}$$

and the corresponding linear programming dual problem as to

$$\underset{\mathbf{u},\mathbf{v}}{\text{maximize}}\ w := \quad \mathbf{b}_2^{\text{T}}\mathbf{u} + v, \tag{3.6a}$$

$$\text{s.t} \quad \mathbf{x}^{(i)\text{T}}\mathbf{B}^{\text{T}}\mathbf{u} + v \leq \mathbf{x}^{(i)\text{T}}\mathbf{c}, \qquad i = 1, \ldots, m \tag{3.6b}$$

$$\mathbf{u} \geq \mathbf{0}^m, \tag{3.6c}$$

$$v \text{ free} \tag{3.6d}$$

Denoting the optimal solution to (3.5) as z^*_{RMP} and the optimal solution to (3.3) as z^*_{MP}, it follows that $z^*_{RMP} \geq z^*_{MP}$, since $\bar{X} \subseteq X$.

The approach of Dantzig-Wolfe decomposition consists of, given an initial feasible solution to the RMP, expanding the solution space of RMP by finding the extreme points of X that, when included in $\bar{X}$, would improve the solution to RMP.

We know that the reduced cost of a new extreme point, $\mathbf{x}^{(m+1)}$ can be expressed in terms of the dual variable values $\mathbf{u}^{(m)}$ and $v^{(m)}$ from the solution in the previous iteration m, respectively, corresponding to the optimal solution of RMP, as

$$\hat{c}_{m+1} := \left(\mathbf{c}^{\mathrm{T}} - \mathbf{u}^{(m)\mathrm{T}}\mathbf{B}\right)\mathbf{x}^{(m+1)} - v^{(m)}. \tag{3.7}$$

The subproblem then consists of finding the extreme point $\mathbf{x}^{(m+1)}$ of X that yields the lowest reduced cost to the given solution, namely the solution to the problem to

$$\underset{\mathbf{x}\in X}{\text{minimize}}\ \hat{c}_{m+1} := \left(\mathbf{c}^{\mathrm{T}} - \mathbf{u}^{(m)\mathrm{T}}\mathbf{B}\right)\mathbf{x} - v^{(m)}. \tag{3.8}$$

If the optimal solution to (3.8) yields a reduced cost $\hat{c}_{m+1} < 0$ we will get a better solution to the RMP if we include the corresponding extreme point $\bar{\mathbf{x}}^{(m+1)}$ in $\bar{X}$ and a corresponding variable λ_{m+1} and resolve the RMP.

The expansion of the RMP will continue as long as the optimal value of the subproblem (the reduced cost) is negative. When the reduced cost of the variable λ_{m+1} which is included in the RMP is greater than or equal to zero we have found the optimal solution to the master problem as there is no other point in X that would improve the solution of the RMP if included in $\bar{X}$.

If matrix $\mathbf{A}$ is block-diagonal, and expressing vector $\mathbf{b}_1$ as

$$\mathbf{b}_1 = \begin{pmatrix} \mathbf{b}_{11} \\ \mathbf{b}_{12} \\ \vdots \\ \mathbf{b}_{1J} \end{pmatrix}, \tag{3.9}$$

where $\mathbf{A}_j \in \mathbb{R}^{m_{1j}\times n_j}$ and $\mathbf{b}_{1j} \in \mathbb{R}^{m_{1j}}$, we define $X_j := \{\mathbf{x}_j \geq \mathbf{0}^{n_j} : \mathbf{A}_j\mathbf{x}_j \leq \mathbf{b}_{1j}\}$. Analogously to the reasoning above, one can express $\mathbf{x}_j$ as a convex combination of the extreme points, $\mathbf{x}_j^{(i)}, i = 1, ..., N_j$, defining the convex hull of X_j. Also, denote $\mathbf{B} = (\mathbf{B}_1, \mathbf{B}_2, ..., \mathbf{B}_J)$, where $\mathbf{B}_j \in \mathbb{R}^{m_2\times n_j}$.

Using these expressions one can formulate the original problem (3.1) as to

$$\begin{aligned}
\underset{\mathbf{x}_i}{\text{minimize}}\ z_{\text{RMP}} = \quad & \mathbf{c}^{\mathrm{T}} \sum_{j=1}^{J}\sum_{i=1}^{N_j} \lambda_{ji}\mathbf{x}_j^{(i)} \\
\text{s.t} \quad & \sum_{j=1}^{J}\sum_{i=1}^{N_j} \lambda_{ji}\mathbf{B}_j\mathbf{x}_j^{(i)} \leq \mathbf{b}_2, && \text{(3.10a)} \\
& \sum_{i=1}^{N_j} \lambda_{ji} = 1, \quad j = 1, \dots J, && \text{(3.10b)} \\
& \lambda_{ji} \geq 0, \quad i = 1, \dots, N_j, j = 1, \dots, J.
\end{aligned}$$

The expression in (3.10) allows for a separation of the subproblem into J smaller subproblems to

$$\underset{\mathbf{x}_j \in X_j}{\text{minimize}} \ \hat{c}_{m+1} := (\mathbf{c}^{\mathrm{T}} - \bar{\mathbf{u}}^{(m)\mathrm{T}}\mathbf{B_j})\mathbf{x}_j - \bar{v}^{(m)}. \tag{3.11}$$

One can then proceed by solving the subproblems individually (e.g. in parallell) in order to obtain extreme points to the sets $X_j, j = 1, \ldots, J$, expand the RMP, and finally obtain an optimal solution to the master problem.

3.2 Applying Dantzig-Wolfe decomposition to the tail assignment problem

To apply Dantzig-Wolfe decomposition to the tail assignment problem as modelled in (2.5), we relax the integer constraints on x_{ijt} which results in the problem to

$$\begin{aligned}
\underset{x_{ijt}}{\text{minimize}} \quad & z := \sum_{j\in I}\sum_{i\in I}\sum_{t\in T} c_{ijt}x_{ijt} && && \text{(3.12a)}\\
\text{s.t.} \quad & \sum_{t\in T}\sum_{j\in I} x_{ijt} = 1, && i \in I\backslash\{i_0, i_N\}, && \text{(3.12b)}\\
& \sum_{j\in I} x_{jit} - \sum_{j\in I} x_{ijt} = 0, && t \in T, i \in I\backslash\{i_0, i_N\}, && \text{(3.12c)}\\
& x_{ijt} \leq e_{ijt}, && i \in I, j \in I, t \in T, && \text{(3.12d)}\\
& r_{jmt} = s_{jmt} + \sum_{i\in I}(1 - v_{ijmt})r_{imt}x_{ijt}, && j \in I, t \in T, m \in M, && \text{(3.12e)}\\
& r_{0mt} = s_{0mt}, && t \in T, m \in M, && \text{(3.12f)}\\
& r_{imt} \leq \beta_{mt}, && i \ \in I, m \in M, t \in T, && \text{(3.12g)}\\
& x_{ijt} \geq 0, && i, j \in I, t \in T. && \text{(3.12h)}
\end{aligned}$$

We let constraints (3.12b) and (3.12h) be the constraints of the master problem, and the other constraints define the subproblem, and therefore get the master problem as to

$$\begin{aligned}
\underset{x_{ijt}}{\text{minimize}} \quad & z_{MP} = \sum_{j\in I}\sum_{i\in I}\sum_{t\in T} c_{ijt}x_{ijt} && && \text{(3.13a)}\\
\text{s.t.} \quad & \sum_{t\in T}\sum_{j\in I} x_{ijt} = 1, && i \in I\backslash\{i_0, i_N\}, && \text{(3.13b)}\\
& x_{ijt} \geq 0, && i \in I, j \in I, t \in T. && \text{(3.13c)}
\end{aligned}$$

Using the path variables defined in Section 2.3.1 and further defining the cost

$\tilde{c}_p = \sum_{t\in T}\sum_{i\in I}\sum_{j\in I}\delta_{ijtp}c_{ijt}$, we can write the master problem as to

$$\begin{aligned}
\underset{y_p}{\text{minimize}} \quad & z_{MP} = \sum_{p\in P}\tilde{c}_p y_p && && (3.14a)\\
\text{s.t.} \quad & \sum_{t\in T}\sum_{j\in I}\sum_{p\in P}\delta_{ijtp}y_p = 1, && i \in I\backslash\{i_0, i_N\}, && (3.14b)\\
& y_p \geq 0, && p \in P. && (3.14c)
\end{aligned}$$

Defining constants $\Psi_{ip} = \sum_{t\in T}\sum_{j\in I}\delta_{ijtp} \forall i \in I, p \in P$, we can rewrite the master problem as to

$$\begin{aligned}
\underset{y_p}{\text{minimize}} \quad & z_{MP} = \sum_{p\in P}\tilde{c}_p y_p && && (3.15a)\\
\text{s.t.} \quad & \sum_{p\in P}\Psi_{ip}y_p = 1, && i \in I\backslash\{i_0, i_N\}, && (3.15b)\\
& y_p \geq 0, && p \in P. && (3.15c)
\end{aligned}$$

The constants Ψ_{ip} equal one if node i is present in path p for aircraft t, and the constraints (3.15b) demands that each node be present in exactly one path in the solution. $y_p \geq 0$ follows directly from $x_{ijt} \geq 0$. We note here that, due to the presence of aircraft nodes in the flight network, knowing a path p, we also know which aircraft node this path went through. The indices t on δ_{ijtp} are thereby redundant, but kept for the sake of clarity. Further note that since only arc costs c_{ijt} are present in the original model (3.13), the cost $\tilde{c}_p$ is uniquely determined by p. The mater problem (3.15) is a set partitioning problem with linearly relaxed decision variables ([23]).

The restricted master problem is defined as

$$\begin{aligned}
\underset{y_p}{\text{minimize}} \quad & z_{MP} = \sum_{p\in \hat{P}}\tilde{c}_p y_p && && (3.16a)\\
\text{s.t.} \quad & \sum_{p\in \hat{P}}\Psi_{ip}y_p = 1, && i \in I\backslash\{i_0, i_N\}, && (3.16b)\\
& y_p \geq 0, && p \in \hat{P}, && (3.16c)
\end{aligned}$$

where $\hat{P} \subseteq P$.

The solution space for subproblem t will be defined as every $\mathbf{x} \in [0,1]^{|I|\times|I|\times|T|}$

satisfying

$$\sum_{j\in I} x_{jit} - \sum_{j\in I} x_{ijt} = 0, \qquad i \in I\backslash\{i_0, i_N\} \tag{3.17a}$$

$$x_{ijt} \leq e_{ijt}, \qquad i \in I, j \in I \tag{3.17b}$$

$$r_{imt} \leq \beta_{mt}, \qquad i \in I, m \in M \tag{3.17c}$$

$$r_{jmt} = s_{jmt} + \sum_{i\in I}(1 - v_{ijmt}) r_{imt} x_{ijt}, \qquad j \in I, m \in M, \tag{3.17d}$$

$$r_{0mt} = s_{0mt}, \qquad m \in M, \tag{3.17e}$$

$$x_{ijt} \geq 0, \qquad i, j \in I \tag{3.17f}$$

$$x_{ijk} = 0, \qquad k \neq t, i \in I, j \in I. \tag{3.17g}$$

The solution space for subproblem t will be denoted as X_t. The subproblem for aircraft t will then be

$$\underset{\mathbf{x}\in X_t}{\text{minimize}} \sum_{i\in I}\sum_{j\in I}(c_{ijt} - u_i^{(m)})x_{ijt}, \tag{3.18}$$

where $u_i^{(m)}$ is the value of the dual variable corresponding to constraint i in (3.15b) the m-th time the master problem has been solved. This subproblem can be characterized as a resource constrained shortest path problem [7, Chapter 2]. The solution to subproblem t will be a flight path p for aircraft t, a new column of values $\Psi_{ip}, i \in I$. To solve this problem we will use a dynamic programming algorithm proposed in [13, Chapter 5, Section 4]. The algorithm is proven to find the optimal solution to the subproblem. This algorithm is described in detail in Section 4.1.1.

Due to the structure of the network of nodes and arcs, every solution to subproblem t must contain aircraft node t. For this reason we know that if we rewrite any solution $\mathbf{x} \in X_t$ to subproblem t, in terms of Ψ_{ip}, then Ψ_{tp} must be equal to one and therefore every column $\Psi_{ip}, i \in I$ is unique and we have a one to one relationship between any path variable y_p and the same solution expressed with variables x_{ijt}.

3.3 Integer quality column generation

A general property of the set partitioning problems that causes high computational complexity is the integer requirement on the variables. In an optimal solution to the MP (3.15), most commonly, very few variables take values close to one (see [13, Chapter 7]). Thereby, after solving the linear relaxation of the tail assignment problem, we will have to take additional actions, such as employing a branch-and-price algorithm [7, Chapter 4, Section 3.1.2] or heuristics, to receive a feasible solution to the tail assignment problem. The branch-and-price approach is complicated to implement in our setting and is likely too computationally demanding for our purposes. We therefore chose to employ a number of heuristic methods to get an integer solution; see Section 4.2.

In recent years, several new methods for finding integer solutions to set partitioning problems using column generation have been proposed. One such method is the *all-integer column generation* [20], by Rönnberg and Larsson in 2014. Another

approach, which we will look into in more detail, is the *integer quality column generation* [6], by Bredström et al. in 2014. The general idea proposed in this paper is to relax the set partitioning constraints and to generate columns, which together with the current solution, minimize the under- and over-coverage of the set partitioning constraints, if they were to be added to the problem. In our case undercoverage is if a flight is not represented in a column in the solution, and overcoverage is if it is represented more than once. This is accomplished by adding a 'surrogate constraint' to the set partitioning problem which is kept as a constraint while the set partitioning constraints are Lagrangian-relaxed ([17, Chapter 6]). The Lagrange multipliers are then updated using a subgradient method, which can be used due to the fact that the Lagrangean dual problem is convex.

In this section, J is the set of columns, I is the set of rows, x_j is the binary decision variable corresponding to column $j \in J$, c_j the cost corresponding to variable x_j, and a_{ij} the entry at row i and column j in the constraint matrix. The linear relaxation of the set partitioning problem with this additional surrogate constraint is to

$$\begin{aligned}
\underset{\mathbf{x}}{\text{minimize}} \quad & \sum_{j\in J} c_j x_j, && && \text{(3.19a)}\\
\text{s.t.} \quad & \sum_{j\in J} a_{ij} x_j = 1, && i \in I, && \text{(3.19b)}\\
& \sum_{j\in J}\sum_{i\in I} a_{ij} x_j = |I|, && && \text{(3.19c)}\\
& x_j \geq 0, && j \in J. && \text{(3.19d)}
\end{aligned}$$

Performing a Lagrangian relaxation of the constraints (3.19b), after a slight rewriting we arrive at the Lagrangian relaxed integer quality optimization problem (LRIQ) to

$$\begin{aligned}
\zeta(\boldsymbol{\gamma}) := & \sum_{i\in I} \gamma_i + \min_{\mathbf{x}} \sum_{j\in J} \left(c_j - \sum_{i\in I} \gamma_i a_{ij} \right) x_j, && && \text{(3.20a)}\\
\text{s.t.} \quad & \sum_{j\in J} \left(\sum_{i\in I} a_{ij} \right) x_j = |I|, && && \text{(3.20b)}\\
& x_j \geq 0, \ j \in J. && && \text{(3.20c)}
\end{aligned}$$

We note that for given values of the multipliers $\boldsymbol{\gamma} \in \mathbb{R}^{|I|}$, there exists a $j^* \in J$ such that

$$\frac{c_{j^*} - \sum\limits_{i\in I} \gamma_i a_{ij^*}}{\sum\limits_{i\in I} a_{ij^*}} \leq \frac{c_j - \sum\limits_{i\in I} \gamma_i a_{ij}}{\sum\limits_{i\in I} a_{ij}}, \quad j \in J. \tag{3.21}$$

The minimization problem in (3.20) for fixed values of the multipliers $\boldsymbol{\gamma}$ is therefore solved by any $\mathbf{x} \in \mathbb{R}^{|I|}$ satisfying

$$x_j = \begin{cases} \frac{|I|}{\sum\limits_{i\in I} a_{ij^*}}, & j = j^* \\ 0, & j \in J \backslash \{j^*\}. \end{cases} \tag{3.22}$$

The linear programming dual problem to the minimization problem in (3.20) is given by the problem (LRIQ-DUAL) to

$$\underset{\delta}{\text{maximize}} \quad \sum_{i \in I} (\gamma_i + \delta) \tag{3.23a}$$

$$\text{s.t.} \quad \sum_{i \in I} (\gamma_i + \delta) a_{ij} \leq c_j, \quad j \in J \tag{3.23b}$$

which is the dual of the lagrangian relaxation of problem (3.19). For a fixed $\boldsymbol{\gamma}$, to solve this maximization problem for δ is simply to find the largest δ which satisfies the constraints (3.23b). One can easily rewrite these constraints as

$$\delta \leq \frac{c_j - \sum_{i \in I} \gamma_i a_{ij}}{\sum_{i \in I} a_{ij}} \quad j \in J. \tag{3.24}$$

From the definition (3.21), one realizes that the value of δ which maximizes (3.23) is found as:

$$\delta^* = \frac{c_{j^*} - \sum_{i \in I} \gamma_i a_{ij^*}}{\sum_{i \in I} a_{ij^*}}. \tag{3.25}$$

The aim of the integer quality column generation is to (by solving the subproblem repeatedly) generate columns which define an optimal integer solution by heuristically controlling the values of the Lagrange multipliers using the subgradient method (Algorithm 3.2) explained below. Hopefully, by controlling the values of the multipliers, $\boldsymbol{\gamma}$, the generated columns will contain a good integer solution to the tail assignment problem. If this is not the case, the method may be combined with the heuristic methods described in Section 4.2, or with a branch-and-bound approach. Pseudo-code for the complete integer quality column generation algorithm is given in Algorithm 3.1, where J_0 is the initial set of columns, T_{init} is the initial number of steps in the subgradient method, $T_{\max}$ is the maximum number of steps and T_{add} is the number of steps added if no column with a negative reduced cost can be found. α is a parameter which controls the step size in the subgradient method. $\bar{c}_{j_0}$ is the least reduced cost found when solving the subproblem. In Algorithm 3.2, $\mathbf{A}$ is the constraint matrix containing the values $a_{ij}, i \in I, j \in J$.

Algorithm 3.1: Integer quality column generation

Input parameters: J_0, T_{init}, $T_{\max}$, T_{add}, α
$\boldsymbol{\gamma} \leftarrow \mathbf{0}$
$p \leftarrow 0$
$T \leftarrow T_{\text{init}}$
$\bar{c}_{j_0} \leftarrow -\infty$
while $T < T_{\max}$ or $\bar{c}_{j_0} < 0$
  $j^* \leftarrow \arg\min_{j \in J_p} \left\{ \frac{c_j - \sum_{i \in I} \gamma_i a_{ij}}{\sum_{i \in I} a_{ij}} \right\}$
  $\delta_p = \frac{c_{j^*} - \sum_{i \in I} \gamma_i a_{ij^*}}{\sum_{i \in I} a_{ij^*}}$
  $\bar{c}_{j_0} \leftarrow \min_{j \in J} \{c_j - \sum_{i \in I} (\gamma_i + \delta_p) a_{ij}\}$
  if $\bar{c}_{j_0} < 0$
    $J_{p+1} \leftarrow J_p \cup \{j_0\}$
  end if
  $(\boldsymbol{\gamma}, T) \leftarrow$ SubgradientMethod($\bar{c}_{j_0}$, $\mathbf{c}$, $\mathbf{A}$, $\boldsymbol{\gamma}$, T, α, J_{p+1})
  $p \leftarrow p + 1$
end while

Algorithm 3.2: Subgradient method

Input parameters: $\bar{c}_{j_0}$, $\mathbf{c}$, $\mathbf{A}$, $\boldsymbol{\gamma}$, T, $\alpha > 0$, J_{p+1}
if $\bar{c}_{j_0} < 0$
  $T \leftarrow T_{\text{init}}$
else
  $T \leftarrow T + T_{\text{add}}$
end if
for t = 1,...,T
  $\mathbf{x} \leftarrow$ Solve LRIQ($\boldsymbol{\gamma}$, $\mathbf{c}$, $\mathbf{A}$)
  for i = 1,...,I
    $G_i \leftarrow 1 - \sum_{j \in J_{p+1}} a_{ij} x_j(\gamma)$
    $\gamma_i \leftarrow \gamma_i + \frac{\alpha}{t} G_i$
  end for
end for
return $(\boldsymbol{\gamma}, T)$

3.3.1 Updating the multipliers: subgradient method

The problem of finding good values of the multipliers, $\boldsymbol{\gamma}$ is solved using the subgradient method ([17, Chapter 6.4]). Controlling the multipliers is done by repeatedly solving the minimization problem in (3.20) given the current values of $\boldsymbol{\gamma}$ to find the column corresponding to the optimal solution $\mathbf{x}$, which is such that $x_{j^*} > 0$ and $x_j = 0$, $j \neq j^*$, as given by (3.21) and (3.22). Then, each element γ_i is updated

based on whether its corresponding flight i is included in column j^*.

Looking at Algorithm 3.2 we see that it is only the variable x_{j^*} that effects how the multipliers are updated, as $x_j = 0$ for all columns except j^*.

If flight i is not included in column j^* we will get G_i equal to one (line 10 in Algorithm 3.2) and thus γ_i will increase by α/t, where α/t is the steplength, and α constant greater than zero. As the objective function to the subproblem of finding a new column is defined as $z = c_j - \sum_{i \in I}(\gamma_i + \delta_p)a_{ij}$ (Algorithm 3.1), the change of γ_i by α/t will contribute to a lower reduced cost for all columns generated that includes flight i, compared to if γ_i would have been unchanged.

If, in subgradient iteration t, flight i is included in column j, G_i will be set to $1 - \sum_{j \in J} a_{ij}/|I|$ and γ_i will be changed by $\alpha/t\big(1 - |I|/\sum_{j \in J} a_{ij}\big)$. As $|I|/\sum_{j \in J} a_{ij}$ is greater than one, γ_i will decrease. This will affect the column generation in such a way that columns including flight i will be less probable to occur as they will have a greater reduced cost compared to if γ_i remained unchanged.

After $\boldsymbol{\gamma}$ is updated, t is increased by one and (3.20) is solved again, using the new value of $\boldsymbol{\gamma}$, which yields a new j^*. The new column j^* is now the column that affects how $\boldsymbol{\gamma}$ is updated in iteration $t + 1$. However, as t is increased by one the step length α/t is smaller and therefore the impact on $\boldsymbol{\gamma}$ from this column will be smaller compared to the previous one.

In conclusion, the change of the element γ_i whose corresponding flight is not represented in any column j^* for any iteration t will be positive. The change of elements whose corresponding flights was represented in a solution j^* for a small value of t will more likely be negative. If the corresponding flight of an element was represented in a column j^* for a large value of t then the difference will at least be small.

Which column is optimal in (3.20) when it is solved in iteration $t = 1$ is decided partly by the value of $\boldsymbol{\gamma}$ from the last time the subgradient update method was used (the last column generation iteration). One can view this as that the column chosen is the one with the best combination of low cost in the RMP and how well it fits with what is over/under-covered (see Section 3.3) according to $\boldsymbol{\gamma}$. As this is the column that fits best integrally (i.e. has the smallest over/under-coverage), it will be the column that have the biggest impact on how $\boldsymbol{\gamma}$ changes in the current subgradient update.

The number of times the multipliers are updated are dependent on the sign of the reduced cost obtained from the subproblem. If the reduced cost is greater than or equal to zero, then no column was added in the previous column generation call. As a result, the number of subgradient iterations T is increased, so that $\boldsymbol{\gamma}$ is updated additional times. According to [6] this makes $\boldsymbol{\gamma}$ converge towards the optimal value of the dual variables in the dual problem of (3.19). However, one wants to keep the number of subgradient iterations low before solving a new subproblem. This is due to the fact that the columns then constructed are more dependent on the over/under coverage of the recently generated columns and therefore are more likely to integrally fit with them. That is, if we solve the RMP using the columns the variables corresponding to the recently generated columns are more likely to assume values close to one compared to columns generated earlier.

4

Solution course for the tail assignment problem

An outline for our solution scheme is sketched in Section 4.1.

A label pulling algorithm for solving the resource constrained shortest path problem (the subproblem in the column generation phase) is presented in Section 4.1.1, in which we also discuss pros and cons of said algorithm.

In Section 4.2, we discuss heuristic methods for obtaining integer solutions to the linearly relaxed problem. The heuristic methods discussed are a connection fixing heuristic by Grönqvist [13] (presented in Section 4.2.2) and an aircraft legality fixing heuristic developed withing our thesis project (presented in Section 4.2.3).

A way to calculate a lower bound on the objective value to the master problem by solving a minimum cost network flow problem is described in Section 4.3. We present the model used for calculating a lower bound from [13], and describe how to set up the network so that the lower bound can be solved quickly using commercial network flow solvers.

Pre-processing methods for reducing the number of arcs in the network defining the subproblem in the column generation step can be found in Section 4.4. The methods presented are guaranteed not to remove the optimal solution to the tail assignment problem from the solution space, if a solution where each flight is covered by an aircraft is possible. Some of the pre-processing methods presented in Section 4.4 are taken from [14], while others are (to the best of our knowledge) presented first in this thesis.

As airlines might want to utilize aircraft differently, due to for example different ages of aircraft, or that a specific aircraft type can be more or less suitable for certain flights, the development performed within this thesis project is made such that the algorithm can handle aircraft-specific requirements. This includes the possibility to have different costs as well as different turn-around times and flight restrictions associated with different aircraft.

4.1 The general approach

There are three major methods used in Algorithm 4.1: the column generation algorithm, the methods used to solve the linearly relaxed restricted master problem, and heuristics used to enforce integer solutions. The main idea is that column generation will generate the columns yielding the lowest reduced cost to incorporate in the linearly relaxed restricted master problem (4.2).

This problem (3.15) can either be the usual linearly relaxed set partitioning problem (see Section 3.2), or it can be the LRIQ-problem detailed in Section 3.3.

As columns are generated we restrict the solution space using two different solution space restricting heuristics. As it would be inefficient to restrict the solution space after each column generation, we let the algorithm generate sufficiently many columns before the solution space is restricted. This is either a given cap of columns or until the least reduced cost of any column generated is no longer negative.

The algorithm continues in this iterative manner until both the solution to the linearly relaxed restricted master problem is integer, and the least reduced cost of any column generated is greater than or equal to zero.

The SolveRRMP method used in Algorithm 4.1 is a method which, given a constraint matrix and the cost of each column in this matrix, outputs the solution to the RRMP (3.15).

The ColumnGeneration method takes the dual variables (or multipliers in case IQ-column generation is used) and the network of nodes as input, and after running Algorithm 4.2, outputs new columns, the cost of the new columns and the reduced cost of the new columns.

The SolutionSpaceRestrictingHeuristic is a method which takes the constraint matrix, the primal variables and the corresponding costs of these variables, and restricts the solution space of the subproblem in accordance to one of the methods described in Section 4.2. This method outputs the restricted network, and the constraint matrix with the columns (paths) which violates the restrictions removed.

4.1.1 A label pulling algorithm for solving the resource constrained shortest path problem

We will use a label pulling algorithm (see [7]) to solve the resource constrained shortest path problem. A label is in our context an object with information of a path, the reduced cost of this path, and the consumption of each resource on this path. A label pulling algorithm is a dynamic programming method, that iteratively finds the shortest path from a given node (in our case, the source node) to each node in the network, subject to a number of resource constraints. The resource constraints are in our case that we have to make sure that (if needed) maintenance can be performed somewhere on the generated route. Since rules are such that certain maintenance has to be performed after every a flying hours, every b departures etc., what we mean by a resource here is for example the number of flying hours since last maintenance check[1]. We must therefore keep track of the value of each resource in every node visited, and make sure that those values do not exceed their corresponding limits β_{mt}.

The essence of the label pulling algorithm is that we iterate through the nodes, and save the paths leading to each node which are *efficient*. To explain the notion of efficient labels, we also need to introduce the concept of *dominance*. Denoting

[1]One could also define other resources to restrict the solution space. For example, if we do not want aircraft to fly more than four flights each day, we could define a resource as *the number of flights taken by an aircraft a given day*, and set its limit to four.

Algorithm 4.1: Pseudocode of the main algorithm

```
1  S − constraint matrix of the master problem,
2      initially set as an identity matrix
3  c − cost of columns in S
4  c − cost of column s
5  X − node network
6  s − set of columns obtained from the column generation
7  r_c − reduced cost
8  π − values of dual variables obtained when solving the RRMP
9  x − values of primal variables obtained when solving the RRMP
10 Parameters:
11 i_initial, initially set to something small
12 i_fix, set to something reasonable
13
14 for i = 1 : i_initial
15    {x, π} ← SolveRRMP(S, c)
16    (r_c, s) ← ColumnGeneration(π, X)
17
18    if (r_c < 0)
19        S ← [S, s]
20        c ← [c, c]
21    else
22        break
23    end if
24 end for
25
26 {x, π} ← SolveRRMP(S, c)
27
28 while ( x is not integer & r_c < 0 )
29    for i = 1 : i_fix
30        {x, π} ← SolveRRMP(S, c)
31        (r_c, s, c) ← ColumnGeneration(π, X)
32
33        if (r_c < 0)
34            S ← [S, s]
35            c ← [c, c]
36        else
37            break
38        end if
39    end for
40
41    {x, π} ← SolveRRMP(S, c)
42    {S, X} ← SolutionSpaceRestrictingHeuristics(S, x, c)
43 end while
44
45 return S
```

the reduced cost of label l as $\bar{c}_l$ and the amount of resource k consumed on label l as r_l^k, we define the notion of dominance as follows:

Definition 1. A label l *dominates* label p if $\bar{c}_l \leq \bar{c}_p$, and $r_l^k \leq r_p^k$ for each resource $k \in M$. □

Definition 2. A label l on node V is *efficient* if it is not dominated by any other label on V. □

Only saving efficient labels on every node is a good strategy since no optimal path for the resource constrained shortest path problem can be dominated [16]. It is however the case that the number of efficient labels on a given node can be huge, as can be seen in Example 4.1.1.

Example 4.1.1. Consider a node V in a network, a resource constrained shortest path problem with one resource, and a label l on V with $\bar{c}_l = M$ and resource consumed $r_V^1 = N$. Then every label p on V with reduced cost $\bar{c}_p = M + n\alpha$ and resource consumed $r_p^1 = N - n\beta$, where $n \in \mathbb{R}$, and $\alpha, \beta \geq 0$, will be efficient. □

Although Example 4.1.1 is constructed so that the number of efficient labels on a node can be made arbitrarily great, it demonstrates that this can be done with just a single resource. In reality, if we have more than one resource, the number of efficient labels on a node can be huge. For computational gain, we thus need a way to limit the number of labels per node, and for that reason we introduce the concept of lexicographical sorting.

Definition 3. A label l is lexicographically less than label p if $\bar{c}_l < \bar{c}_p$, or if $\bar{c}_l = \bar{c}_p$ and $\exists q : r_l^q < r_p^q$ and $r_l^k = r_p^k,\ k \in [0, q)$.

Using lexicographical sorting, we assume that the resources are initially sorted according to their importance. The importance here can be chosen arbitrarily, or according to how hard the resource constraints are to satisfy.

We let L_V be the number of labels on a node, and $\underline{\chi}$ and $\bar{\chi}$ (where $\underline{\chi} \leq \bar{\chi}$) be two pre-defined lower and upper bounds on the number of labels in a node. In each node, we will now store labels according to the following procedure:

- If $L_V < \underline{\chi}$, we insert a new label if it is efficient with respect to the already saved labels on V, and remove dominated labels from V after insertion.
- If $\underline{\chi} \leq L_V < \bar{\chi}$, we save a new label if it is efficient and lexicographically less than the greatest label in V. We remove dominated labels after the insertion.
- If $L_V = \bar{\chi}$ after insertion, we remove each dominated label. If no label was dominated after insertion, we remove the lexicographically greatest label, so that $\bar{\chi}$ labels are remaining for node V.

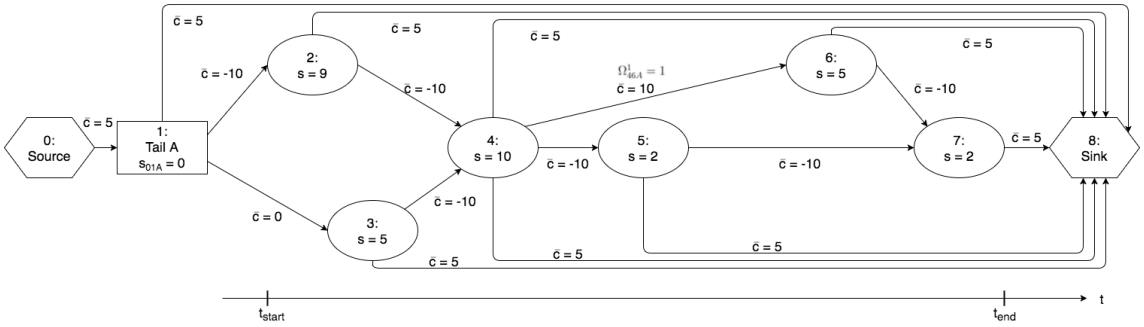


Figure 4.1: A network consisting of one aircraft node and six flight nodes. The reduced cost of using a specific arc is denoted as $\bar{c}$. The resource consumption is denoted as s.

This can be seen on rows 14 to 25 in Algorithm 4.2.

This procedure (Algorithm 4.2) is heuristic, and does not guarantee that the least reduced cost path through the network is found. It is however generally necessary to use this approach, since the number of paths which have to be evaluated is equal to the number of labels in the network. It is also the case that using column generation, we do not need to find the *least* reduced cost solution to the subproblem, but it suffices to find a negative reduced cost solution. If no negative reduced cost solution is found, one could try raising (or removing) the limits $\underline{\chi}$ and $\bar{\chi}$, to make sure that the solution to the RRMP can not be improved further. If the bounds $\underline{\chi}$ and $\bar{\chi}$ are removed, one will inevitably find the shortest path through the network. This is due to the fact that no optimal path can be dominated, and without the bounds $\underline{\chi}$ and $\bar{\chi}$ every efficient label will be stored when labeling the sink node.

What also has to be mentioned (and what makes this subproblem different than a standard resource constrained shortest path problem) is the possibility for maintenance of type m to be performed on aircraft t between flights i and j in the generated path, and thus resetting the resource r_l^m for label l on node j to s_{jmt}. Example 4.1.2 illustrates the concepts of dominance, resetting of resources, and efficiency.

Example 4.1.2. Performing the label pulling algorithm on the network displayed in Figure 4.1, when considering a single resource r_1 with a resource limit $\beta = 20$ for the consumption of that resource, and $\underline{\chi} = 6$ and $\bar{\chi} = 8$, results in the intermediate steps depicted in Table 4.1.

There are unique paths leading to nodes 1 through 3 in the network, and we thereby just set one label each for these nodes. To node 4, there are two paths, which are both efficient since one has a lower reduced cost (-15), and the other has a lower consumption of the resource (15). In node 5, we get a path which has a resource consumption that is higher than the limit 20: this path is thereby infeasible and a label for it will not be set. Note that this infeasibility corresponds to a violation of one of the constraints in (2.5f). Between nodes 4 and 6, $\Omega^1_{46A} = 1$, and maintenance can be performed. The resource consumed on paths leading to node 6 will thereby be reset to $s_6 = 5$. As a result of this, the two paths leading to node 6 will have different costs but the same amount of resource consumed. The label with the lower cost thereby dominates the one with the higher cost, and the

latter one will be removed. On node 7, one label has a lower cost, and the other has a lower consumed resource, so both are efficient. On the sink node, four of the labels are efficient after all labels have been pulled. □

Node	Path	Reduced cost	Resource consumed	Comment
1	{0,1}	5	0	
2	{0,1,2}	-5	9	
3	{0,1,3}	5	5	
4	{0,1,2,4}	-15	19	
4	{0,1,3,4}	-5	15	
5	{0,1,2,4,5}	-25	21	$r > \beta$, infeasible path
5	{0,1,3,4,5}	-15	17	
6	{0,1,2,4,6}	-5	5	Resetting r.
6	{0,1,3,4,6}	5	5	Resetting r. Dominated.
7	{0,1,3,4,5,7}	-25	19	
7	{0,1,2,4,6,7}	-15	7	
8	{0,1,8}	10	0	
8	{0,1,2,8}	0	9	Dominated.
8	{0,1,3,8}	10	5	Dominated.
8	{0,1,2,4,8}	-10	19	Dominated.
8	{0,1,3,4,8}	0	15	Dominated.
8	{0,1,3,4,5,8}	-10	17	Dominated.
8	{0,1,2,4,6,8}	0	5	
8	{0,1,3,4,5,7,8}	-20	19	
8	{0,1,2,4,6,7,8}	-10	7	

Table 4.1: Labels in the intermediate steps when performing the label pulling algorithm on the network depicted in Figure 4.1 considering a single resource r_1.

A downside of using the label pulling algorithm is that no solution is found before the sink node is labeled, and the sink node is the last node to be labeled. Lately, there has been new solution techniques based on branch-and-cut developed for solving the resource constrained shortest path problem [11]. It is, however, unclear if one can implement the possibility to reset resources to fixed values using this method, and thereby it is unclear whether this is a reasonable approach to this specific version of the problem. In this thesis we have not looked further into the technique presented in [11], and consider its possible implementation for the tail assignment problem as a subject for future research. An upside to the presented version of the subproblem, however, is that the network on which it is solved, is a directed acyclic graph. Because of the time dependence concerning which flights can be predecessors to other flights, it is, even with negative edge costs, impossible to get negative cost cycles (which is generally a problem when solving shortest path problems).

In Algorithm 4.2, L_V is the set of labels on node V. $\Omega^m_{V'Vt} \in \{0,1\}$ is equal to one if maintenance of type m can be performed on aircraft t between activities in V' and in V, and zero otherwise. The flight nodes are sorted in the order of departure times, and the labels on each node are lexicographically sorted. Algorithm 4.2 is to

Algorithm 4.2: Label pulling algorithm

```
1  for each node V ∈ I
2    for each predecessor V' of V ∈ I
3      for each label l_V' ∈ L_V'
4        l_V ← new label
5        for each maintenance activity m ∈ M
6          if Ω^m_V'Vt = 1
7            r^m_lV ← s^m_V
8          else
9            r^m_lV ← r^m_lV' + s^m_V + s^m_V'V
10         end if
11       end for
12       c̄_lV ← c̄_lV' + c_V'Vt − (π_V' + π_V)/2
13       c_lV ← c_lV' + c_V'Vt
14       if l_V efficient w.r.t. L_V
15         if |L_V| < χ̲
16           L_V ← L_V ∪ {l_V}
17           remove dominated labels from L_V
18         else if l_V <_lex max_lex(L_V)
19           L_V ← L_V ∪ l_V
20           remove dominated labels from L_V
21           if |L_V| > χ̄
22             Remove lexicographically greatest label from L_V
23           end if
24         end if
25       end if
26     end for
27   end for
28 end for
```

a large extent based on the one developed by Grönkvist in [13].

4.1.2 Fixed activities

Airlines generally want to plan longer maintenance work in advance. These events are present as preassigned (or fixed) activities in our model. To make sure that we generate routes which cover these activities when solving the subproblem, we treat them as nodes which must (and can only) be covered by one specific aircraft. This is done by making sure that, when pulling a label on a flight node departing after a fixed activity, any path related to the label pulled must have passed through the fixed activity if it also passed through the aircraft node corresponding to the aircraft on which maintenance should be performed. Making flight $\tilde{j}$ a fixed activity for aircraft $\tilde{t}$ is accomplished by adjusting the allowed arc flows in two ways. Firstly, the allowed arc flow to node $\tilde{j}$, $e_{i\tilde{j}t}$, is set to zero for every aircraft $t \in T \backslash \{\tilde{t}\}$. Secondly the allowed arc flow $e_{ij\tilde{t}}$ for every arc connecting an activity i arriving before activity $\tilde{j}$ departs, to activity j departing after $\tilde{j}$ departs, is set to zero. Additionally, setting arc flows to zero in this manner can (and most probably will) 'cut off' some flights for aircraft $\tilde{t}$. If there is no way to make a path containing both flights $\tilde{\imath}$ and $\tilde{j}$ for aircraft $\tilde{t}$, every arc flow $e_{i\tilde{j}\tilde{t}}$, $i \in I$, and $e_{\tilde{\imath}j\tilde{t}}$, $j \in J$, can be set to zero.

Note that this way of handling fixed activities is not restricted to preassigned maintenance, but can also be applied to flights which should be assigned to a specific aircraft. We detail this in Section 4.2.3, which describes aircraft fixing heuristics.

4.2 Heuristics

While column generation can find the optimum of the RRMP there is no guarantee that an optimal solution to the RRMP possesses binary variable values. In fact, it seldom does. In order to obtain a feasible integer solution we need to adjust the solution to the RRMP, or the way the solution to the RRMP is created.

The approach behind the heuristic methods presented below is to restrict the solution space of the RRMP and the way columns are allowed to be created, so that they are more likely to have integrality properties while also yielding a low objective value.

The heuristics presented are motivated by the assumption that some properties of a good integer solution to the RRMP will also be properties of the optimal (non-integer) solution to the RRMP. Which properties or parts of the schedule that are looked at are specified in detail in Sections 4.2.1 through 4.2.3 where the different heuristics are explained.

Using heuristics results in a faster algorithm for obtaining an integer solution, as compared to the method of branch-and-price [13]. Furthermore, since the heuristic methods restrict the solution space and thus the number of columns that can be created, they result in faster column generation after they are applied. However, the downside of using heuristic methods is that there is no guarantee that we will end up with an optimal solution, and if too aggressive heuristics are applied, the integer solution found is likely to be far from optimal.

4.2.1 Variable fixing heuristic

The most straightforward heuristic presented in this report is the variable fixing heuristic. It stems from the logic that if the value of a variable in an optimal solution to the RRMP is close to one, then the corresponding path will probably contribute to a good integer solution. This method has been shown to be less effective at producing good integer solutions as compared to other heuristics (see [13]), and therefore we will not present any results from using it. We include it in our presentation merely as a way to motivate the methods presented in Sections 4.2.2 and 4.2.3.

We choose the variables to fix as those satisfying

$$y_p \geq 1 - \epsilon_t, \tag{4.1}$$

where the value if $\epsilon_t > 0$ is initially close to zero. If no variables y_p that fulfill condition (4.1) can be found, then the value of ϵ_t is increased until at least one variable satisfies the condition (4.1).

If the value of ϵ_t becomes greater than 0.5, only one variable is fixed as it is possible that there are two columns i_1, i_2, where $1 - \epsilon_t \leq y_{p_1} \leq 0.5$ and $1 - \epsilon_t \leq y_{p_2} \leq 0.5$, and where the two columns share at least one flight. Both these columns could not be fixed to one, since this would violate constraints (3.15b).

To illustrate how the solution space is restricted we show what happens when a variable is fixed: Given paths $\Psi_{\cdot p}$ in the set of paths (or columns) P, where $\Psi_{ip} = 1$ if flight $i \in I$ is included in path $p \in P$, the costs c_p and the corresponding variables y_p we can formulate the RRMP as to

$$\begin{aligned}
\underset{y_p}{\text{minimize}} \quad & \sum_{p \in P} c_p y_p, && \text{(4.2a)} \\
\text{s.t.} \quad & \sum_{p \in P} \Psi_{ip} y_p = 1, \quad i \in I, && \text{(4.2b)} \\
& y_p \geq 0, \quad p \in P. && \text{(4.2c)}
\end{aligned}$$

If the variable y_{p_0} would be fixed to 1, then the resulting problem is expressed as to

$$\begin{aligned}
\underset{y_p}{\text{minimize}} \quad & \sum_{p \in P} c_p y_p, && \text{(4.3a)} \\
\text{s.t.} \quad & \sum_{p \in P} \Psi_{ip} y_p = 1, \quad i \in I, && \text{(4.3b)} \\
& y_p \geq 0, \quad p \in P, && \text{(4.3c)} \\
& y_{p_0} = 1. && \text{(4.3d)}
\end{aligned}$$

What happens to the flight network when the variable fixing heuristic is applied is illustrated in Figure 4.2. An integer solution is enforced by iteratively creating new paths and fixing more and more variables to 1. When a path for every aircraft

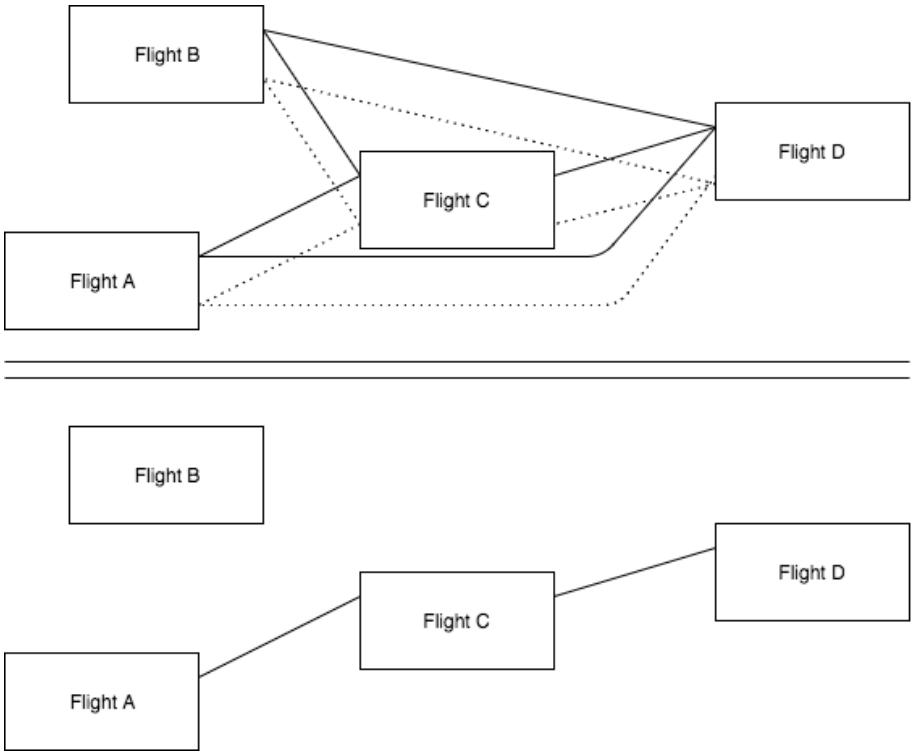


Figure 4.2: Illustration of the variable fixing procedure. The filled and dotted lines represent arcs for two different aircraft. The upper figure shows a part of the flight network before any fixing has occurred. In the lower figure, a path has been made containing the flights A, C and D for the aircraft related to the filled lines. The corresponding variable y_p to this path fulfilled the condition $y_p \geq 1 - \epsilon_t$ and has thereby been fixed to 1. As a result, the aircraft related to the filled lines, and the flights A, C and D, need no longer be considered when solving the subproblem.

has been fixed, the solution to the RRMP will be integer. However we cannot ensure that we get an integer solution that is feasible in the practical sense, as it may be the case that we restrict the solution space so that no integer solution with all flights assigned to aircraft can be found. One example of this is if we have flights A, B and C, where the only predecessor to C is A. If in this situation a column containing A and B is fixed, then no integer solution where C is covered by an aircraft can be found.

4.2.2 Connection fixing heuristic

A less aggressive approach compared to fixing entire paths, as done by the variable fixing, is to fix parts of paths that are included in a ”good” solution to the RRMP as well as close to integrality.

Such parts, or sub paths, can for example be connections between flights. A connection between two flights in a path exists if the latter of the two is directly succeeding the first.

A connection is said to be close to integrality if the sum over the variables corresponding to paths where the connection is occurent is close to either zero or one.

Using the same assumption that motivated the variable fixing heuristic, that parts of a schedule that is close to integrality, when the binary constraint on the variables in the master problem (2.5) is relaxed, will probably be in a good solution to the master problem with binary constraints on the variables, we want to find connections that are close to integrality. As with the variable fixing we want to fix these connections in order to enforce integrality as well as speed up the algorithm.

To fix a connection we remove all outgoing connections from the first flight in the connection except the connection to the last flight in the connection as well as removing all connections to the latter flight's predecessors except the one in the connection at hand; see Figure 4.3 for an illustrative example.

In order to fix connections we need to find which connections that can be used. To do that we define constants σ_{ijp}, as one if flight j is directly following flight i in path p and zero otherwise. Thereafter we sum over all paths and all connections to obtain a so called *connection score*, defined as $c_{ij}^{\text{score}} = \sum_p \sigma_{ijp} y_p$, which is used to evaluate the connections. Here, y_k is a relaxed decision variable in the RRMP i.e., (4.2). Observe that if the variable corresponding to path p, $y_p = 1$ in the RRMP, then the connection score $c_{ij}^{\text{score}} = 1$ as well. Further one realizes that the largest connection score possible is one as the connection score is bounded by the set partitioning constraint in (3.19b). This means that we can evaluate the integrality of the connections by how close to 1 the connection score is.

Analogously to the variable fixing we can then choose to fix connections (i, j) satisfying

$$c_{ij}^{\text{score}} > 1 - \epsilon, \tag{4.4}$$

for some small value of $\epsilon > 0$. Again, if no such connections can be found, the value of ϵ is increased until at least one connection can be found that satisfy condition (4.4). If $\epsilon > 0.5$ only one connection will be fixed; this in order to avoid the possibility that two (or more) connections from the same flight are fixed. A pseudo-code of the algorithm can be found in Algorithm 4.3.

4.2.3 Aircraft legality fixing heuristic

As our model allows each aircraft to have a specific objective function, an individual minimum required ground time between flights, and an individual initial positioning, the cost of assigning a flight to one aircraft compared to another can differ. As a result it matters which aircraft is assigned to which flight when paths are created.

It is therefore reasonable to assume that the assigning of flights to aircraft in the RRMP indicates how flights should be assigned to aircraft in the integer solution in order to obtain a good solution.

If we aim to obtain a schedule without any unassigned flights, each flight will have exactly one aircraft assigned to it. Analogously, as finding flight connections with a high score in the connection fixing heuristic, we may search for aircraft frequently assigned to flights in the RRMP, and fix aircraft to flights if they yield a good solution. To evaluate the assignment of aircraft to flights we define node pairs and a corresponding pair score. A pair, (i, t), consists of a flight node i and an aircraft node t and the corresponding pair score is denoted as ϕ_{it}^{score}.

Algorithm 4.3: Connection fixing heuristic

```
1  ℂ − set of all connections
2  C̃ = {∅}, connections to fix.
3  returnOneConnection ← false
4  Parameters:
5  ϵ ∈ [0.05, 0.15]
6  c_incremental ∈ (1, 1.3]
7
8  while C̃ = ∅
9    ∀ connections c in ℂ
10     if (c^score > 1 − ϵ)
11       if (returnOneConnection = false)
12         C̃ ← C̃ ∪ c
13       else
14         C̃ ← c
15         break for
16       end if
17     end if
18   end for
19
20   ϵ ← ϵ · c_incremental
21   if (ϵ ≥ 0.5)
22     ϵ ← 0.5
23     returnOneConnection ← true
24   end if
25 end while
26 return C̃
```

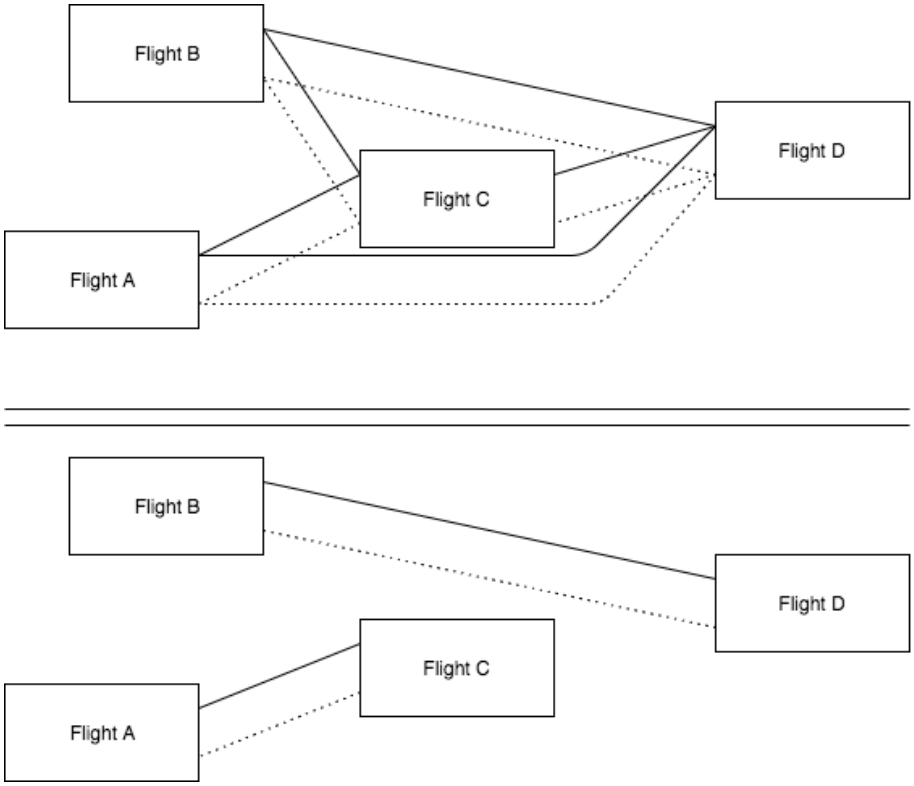


Figure 4.3: The connection fixing procedure. For each connection there are two legalities, the filled and the dotted line. The connection between flight B and flight D is fixed. As a result, all outgoing connections from flight B except for the connection (and its included legalities) to flight D are removed as well, as all incoming connections to flight D except for the connection from flight B are removed. If a path including flight B is created, then flight D must also be included. The size of the problem is decreased as the number of possible ways to create paths is decreased. Further, as there are fewer ways to create paths including flights B and D and therefore the solution to the RRMP is more likely to be integer. However, it is not decided which aircraft should be assigned to the flights.

$$\phi_{it}^{\text{score}} = \sum_{p} y_p \mathbb{1}_{\{\text{flight } i \text{ and aircraft } t \text{ is in path } p\}}, \tag{4.5}$$

where $\mathbb{1}_{\{\text{flight } i \text{ and aircraft } t \text{ is in path } p\}}$ is the indicator function, being one if flight i and aircraft t is in the path corresponding to the variable y_p and zero otherwise.

Pairs with a high pair score in the RRMP are likely to contribute to a good solution to the RRMP, and we assume that they are also likely to contribute to a good solution to the RRMP with binary constraints on the variables. Furthermore one can assume that pairs with a low pair score are less likely to contribute to a good solution.

To find such pairs we have defined an algorithm that shares a lot of similarities with the connection fixing-procedure. We search for pairs by iteratively decreasing an upper bound, $1 - \epsilon$, until we find pairs whose pair score satisfy the inequality $\phi_{it}^{\text{score}} > 1-\epsilon$. The first pairs satisfying this are the ones whose corresponding legality should be fixed, see pseudo-code in Algorithm 4.4.

As with connection fixing there is a risk that we restrict the solution space too aggressively in order to obtain an integer solution.

To compare the aggressiveness between aircraft fixing and the connection fixing heuristics we need to have aircraft specific connections, call them arcs. For each connection between two flight nodes, the number of arcs equals the number of common legal aircraft on the two flight nodes. In the results section we compare the two heuristics with respect to their relative aggressiveness.

A visualization of how the aircraft legality fixing can be seen in Figure 4.4

4.3 Lower bound on the optimal value of the tail assignment problem

In order to evaluate how well the algorithms perform, we need a lower bound on the optimal value of the tail assignment problem (2.5). One way to get such a bound is to relax the problem in such a way that the turn around time (ground time between flights) before every flight is equal to the minimum turn around time possible for that flight. Every maintenance constraint is removed (except for preassigned activities). The cost of using an arc between flight A and flight B in the network is equal to the lowest cost of using any arc between flight A and flight B. Below we define the model used to calculate a lower bound.

Algorithm 4.4: Aircraft legality fixing

```
1   Φ − set of all pairs
2   Φ̃ = {∅}, pairs which corresponding legality should be fixed
3   returnOnePair ← false
4   Parameters:
5   ϵ ∈ [0.05, 0.15]
6   c_incremental ∈ (1, 1.3]
7
8   while Φ̃ = ∅
9     ∀ pairs (i, t) in Φ
10      if ( φ^score ∈ [1 − ϵ, 1] )
11        if ( returnOnlyOnePair = false )
12          Φ̃ ← Φ̃ ∪ (i, t)
13        else if ( returnOnlyOnePair )
14          Φ̃ ← (i, t)
15        end if
16      end
17    end for
18
19    ϵ ← ϵ · c_incremental
20    if (ϵ ≥ 0.5)
21      ϵ ← 0.5
22      returnOnePair ← true
23    end if
24  end while
25  return Φ̃
```

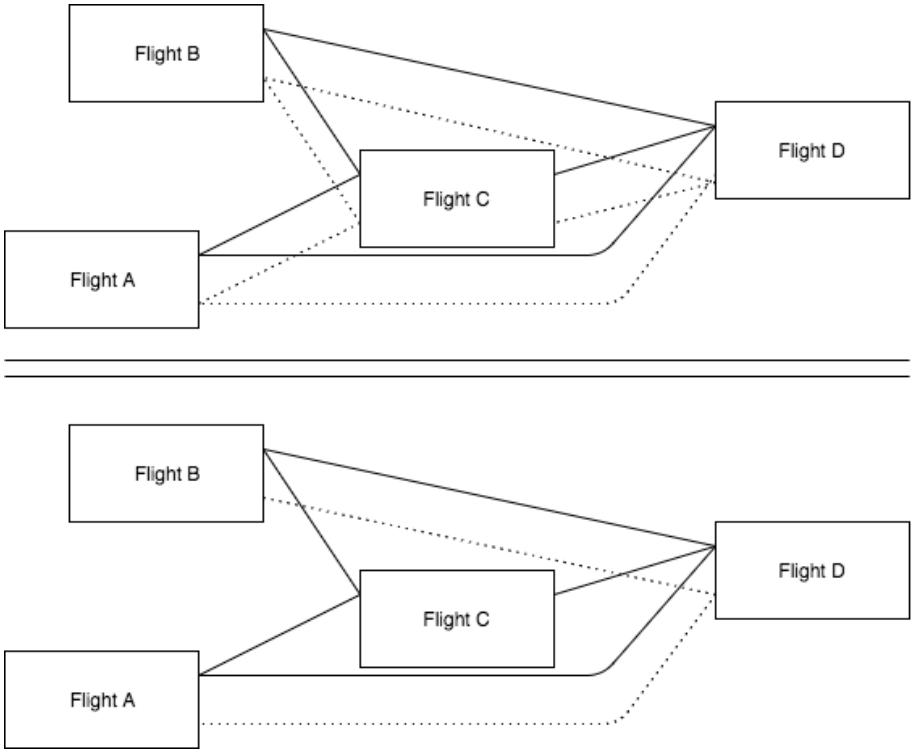


Figure 4.4: Aircraft legality fixing procedure, isolated around flight C and its legality connections. For simplicity the legalities are represented as connections. There are two legalities, represented by filled and dotted lines. The legality (aircraft) represented by the filled line is fixed on flight C and thus every path including flight C must contain the aircraft node represented by the filled connection. The number of possible legality connections have decreased, the problem is restricted, and it is more likely that an integer solution will be obtained from the algorithm.

$$\underset{x_{ij}}{\text{minimize}} \quad \sum_{i\in I}\sum_{j\in I} c_{ij}x_{ij}, \tag{4.6a}$$

$$\text{s.t.} \quad \sum_{i\in I} x_{ij} = 1, \quad j \in I\backslash\{i_0, i_N\}, \tag{4.6b}$$

$$\sum_{j\in I} x_{ij} = 1, \quad i \in I\backslash\{i_0, i_N\}, \tag{4.6c}$$

$$\sum_{i\in I} x_{ii_s} = |I_\text{a}|, \tag{4.6d}$$

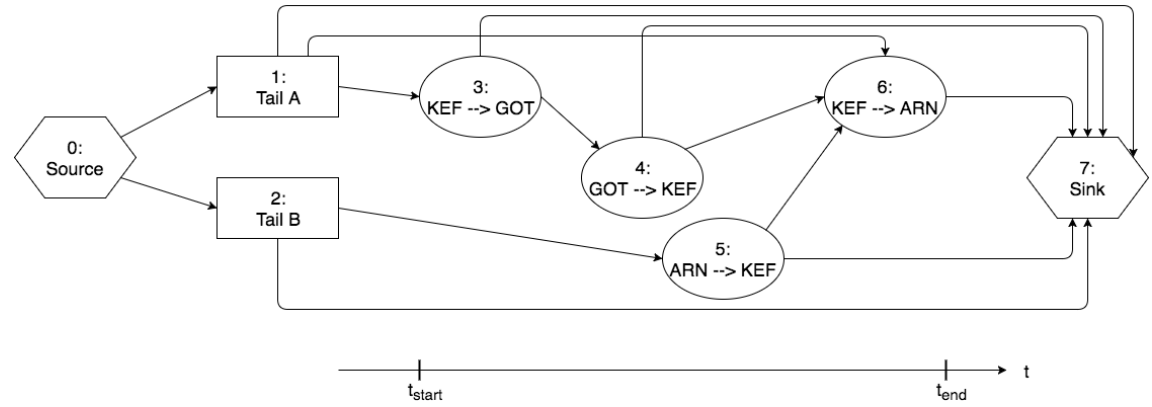
$$\sum_{j\in I} x_{i_0 j} = |I_\text{a}|, \tag{4.6e}$$

$$0 \leq x_{ij} \leq e_{ij}, \qquad i, j \in I. \tag{4.6f}$$

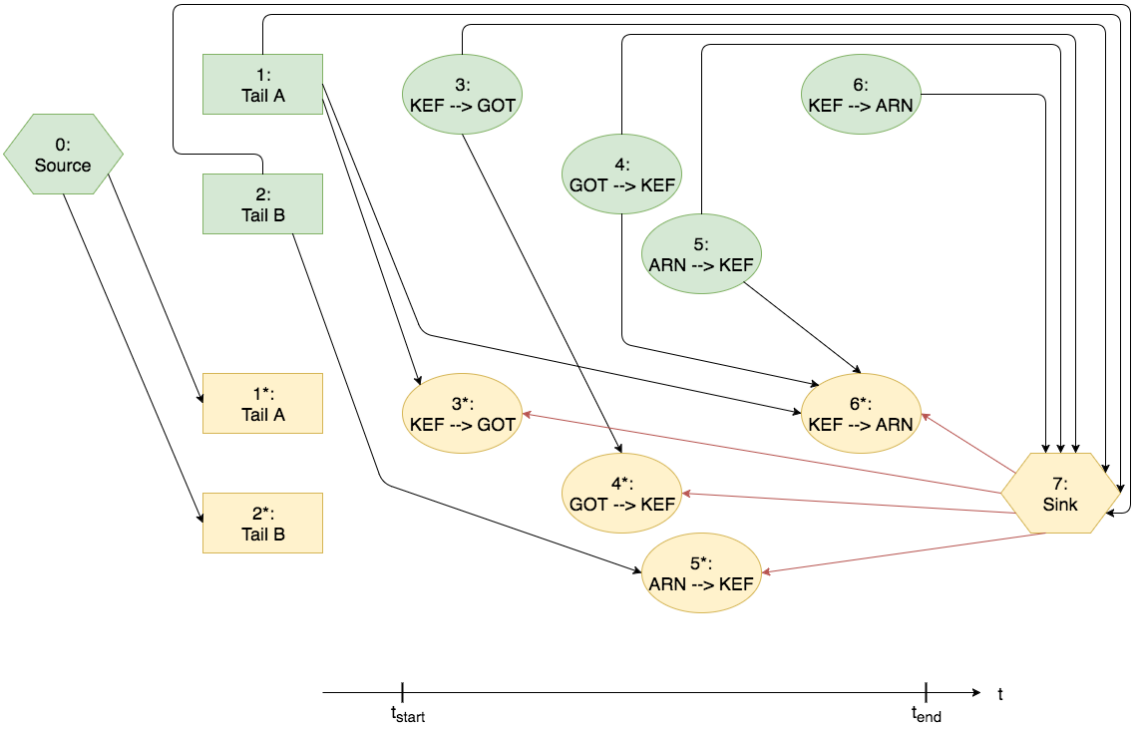
Observe that only two indices are used for x, e and c in this problem formulation, since (4.6) concerns anonymous aircraft. That is, e_{ij} is the flow capacity on arc (i, j), x_{ij} is the binary variable which equals one if arc (i, j) is used in the solution, and c_{ij} is the cost of a unit of flow on said arc. Other than this, the same definitions of constants and variables are used in this model as in (2.5). Model (4.6) is a minimum cost network flow problem and can be solved quickly using a standard solver. Constraints (4.6b) and (4.6c) demand that we have a flow balance exactly equal to one through each node. Since flow balance in a standard minimum cost network flow problem just demands that the flow into a given node is equal to the flow out of that node, these constraints are somewhat complicating when viewing (4.6) as a minimum cost network flow problem. Decomposing the network into a *supply network* and a *demand network*, we can however get around this problem. We will begin to explain this procedure using an example.

Example 4.3.1. Figure 4.5a shows a network consisting of two aircraft nodes, four flight nodes, a source node, and a sink node. Solving the minimum cost network flow problem on this network directly would not consider the constraint that the flow through each node must be equal to one. To enforce this property, we split each aircraft and flight node into a *supply node* which generates one unit of flow, and a *demand node* which demands one unit of flow (see Figure 4.5b). We thereafter redirect the arcs from each supply node to its successor's demand nodes. Further, expensive directed arcs from the sink node to each flight's demand node are added, and using these corresponds to leaving the corresponding flights unassigned.

We note that using this model, we have no control over which particular aircraft covers a preassigned activity, just that *some* aircraft covers it. If we use different objective functions for different aircraft, the lowest arc cost over all objective functions will be used to compute a lower bound on the optimal value of (4.2). If there are flights for which different aircraft have different turn around times, or if there are flights that are not allowed to be assigned to some particular aircraft, this can not be taken into account when calculating the lower bound this way. The lowest turn around time allowed for any aircraft will instead be used when placing the arcs in the network, and the lower bound model will be equivalent to one in which each aircraft is allowed to take any flight. If there are more than one flight which needs



(a)



(b)

Figure 4.5: (a) A small network consisting of two aircraft and four flights. (b) The network in (a) decomposed into two layers. The yellow nodes have a demand of one unit of flow. The green nodes have a supply of one unit of flow. The source has a supply equal to the number of aircraft nodes. The sink node has a demand equal to the number of aircraft nodes. Each arc has a capacity of one unit of flow, and a cost c_{ij}. The sink node has backward arcs to each flight "demand" node, which use corresponds to leaving the respective flight unassigned. Decomposing the network in this way, a minimum cost network flow solver can be applied directly, and a lower bound can be obtained for the tail assignment problem (2.5).

to be left unassigned in the optimal solution to (2.5), the lower bound obtained from solving (4.6) might be quite optimistic (see table 5.1). This can happen, since it is not necessary that one backward arc to each unassigned flight is used in an optimal solution to (4.6). If, for example, flights A and B are unassigned in the optimal solution to (2.5), in the optimal solution to (4.6) there might be a positive flow on the backwards arc to A, and a positive flow on the arc from A to B. Since (4.6) is a minimum cost network flow problem, its constraint matrix is totally unimodular, and an integer solution will be obtained even without integer constraints on the variables [23, Section 3.2]. Due to this, and the reasons stated above, the lower bound will be closer to the optimal value of (2.5) if the same objective function is used for each aircraft, and closer for a uniform fleet than for a mixed fleet.

4.4 Reducing the number of arcs in the network

An important issue when solving the tail assignment problem is the time complexity. The resource constrained shortest path problem is NP-hard [12, Appendix 2], and limiting the solution space to our subproblem is thereby of great benefit. To speed up the solving of the tail assignment problem, we can limit the number of labels allowed on each node. This could have a negative effect on the convergence, since (as discussed in Section 4.1.1) it is possible that the optimal path through the network is not found when the number of labels is limited. The time complexity will however still be exponential as a function of the number of nodes in the network, since adding the $n+1$'th node to a network will result in the addition of αn new arcs, where $\alpha \in \mathbb{Z}_+$. There are, however, a few techniques one can use to remove edges which can not appear in paths describing an optimal solution to the tail assignment problem.

We have used three different methods of reducing the number of arcs, motivated by Lemmata (4.4.2)–(4.4.4) (the first is proposed by Grönkvist and Kärrström [14] and the two latter by us), which can all be derived from the following Lemma that we propose:

Lemma 4.4.1. *Let t_k^{Dep} be the departure time of flight f_k, t_k^{Arr} be the arrival time of flight f_k, and $|I_{\text{A}}|$ be the number of aircraft available. Further suppose we have a subset of flights I_{s}, where t_{start} is the time of the earliest departure for any flight in I_{s} and t_{end} is the latest arrival time for any flight in I_{s}.*

If exactly $|I_{\text{A}}|$ unique aircraft must be utilized to create paths including all flights in I_{s} in order to create a solution to the tail assignment problem without any unassigned flights, then no arcs between the flight nodes f_i and f_j can exist in that solution, where $t_i^{\text{Arr}} < t_{\text{start}}$ and $t_j^{\text{Dep}} > t_{\text{end}}$.

Proof. This is true since if such an arc is used in a solution, then at least one flight, f_k, $f_k \in I_s$, must be left unassigned. □

A special case of Lemma (4.4.1) is discussed in the article [14], where Grönkvist and Kjerrström reason that, if there exists a point in time during the time span under consideration where every available aircraft has to be used, every route in a solution to the tail assignment problem without any unassigned flights must contain exactly one of these flights. This is more precisely stated in Lemma 4.4.2.

Lemma 4.4.2 (Grönkvist and Kärrström (2005))**.** *. Let $F := \{f_i\}_{i=1}^{n}$ be the set of flight nodes under consideration. Let t_i^{Dep} be the departure time of flight f_i and t_i^{Arr} be the arrival time of flight f_i. Let $F_t \subseteq F$ be the set of flights that are in the air at time t (i.e., all flights f_i such that $t_i^{\text{Dep}} < t$ and $t_i^{\text{Arr}} > t$) and let $|I_{\text{A}}|$ be the number of available aircraft.*

If $|F_{t'}| = |I_{\text{A}}|$ for some given time t', and if $t_l^{\text{Dep}} < t'$ and $t_m^{\text{Arr}} > t'$, then no solution without any unassigned flights to the tail assignment problem can contain a path using an arc between f_l and f_m.

Proof. Using an arc between flight nodes f_l and f_m would mean that one of the flights in F_t would be left unassigned. □

From our experience, this happens quite frequently for certain airlines with a lot of traffic, and using this fact the number of arcs in the network (and thereby the solution time for the subproblem) can sometimes be significantly reduced. Note that if $|F_{t'}| > |I_A|$ for some given time t', no solution with all flights covered is possible. If this is the case, removing arcs according to Lemma 4.4.2 might in fact remove the optimal solution from the solution space. As a result, one should apply Lemma 4.4.2 with caution.

Another way to remove arcs from the network without removing feasible solutions to the problem is to look for time spans during which each available aircraft must depart from a specific airport exactly once. If such a time span is found, a direct implication of Lemma 4.4.3 is that we can remove any edge starting from a flight arriving before the time span, to flights departing after the time span. This is quite likely to happen for airlines which have a main 'hub' from/at which most flights depart/arrive.

Lemma 4.4.3. *Let $|I_{\text{A}}|$ be the number of available aircraft, F the set of flights ordered by departure time and F^i the set of flights departing from airport i ordered by departure time. Let $\{f_{i_1}, f_{i_2}, ..., f_{i_{|I_{\text{A}}|}}\} \subseteq F^i$ denote $|I_{\text{A}}|$ subsequent flights in F^i, t_j^{Dep} the departure time of flight j and t_j^{Arr} the arrival time of flight j. Let $f_l \in F$ and $f_m \in F$ be two flights such that $t_l^{\text{Arr}} < t_{i_1}^{\text{Dep}}$ and $t_m^{\text{Dep}} > t_{i_{|I_{\text{A}}|}}^{\text{Arr}}$. If no path can be made in the network containing two or more of the flights in the set $\{f_{i_1}, f_{i_2}, ..., f_{i_{|I_{\text{A}}|}}\}$, then no solution to the tail assignment problem without any unassigned flights can contain a path using an arc between f_l and f_m.*

Proof. Using an arc between flight nodes f_l and f_m would mean that one of the flights in the set $\{f_{i_1}, f_{i_2}, ..., f_{i_{|I_{\text{A}}|}}\}$ would be left unassigned. □

An example case of when Lemmata 4.4.2 and 4.4.3 can be used is presented in Figure 4.6.

Lemma 4.4.4 follows a similar reasoning, where we instead look for time spans during which each available aircraft must *arrive* at a specific airport exactly once. Although one might initially be inclined to believe so, Lemma (4.4.2) and (4.4.3) are not equivalent. An example of this can be seen when inspecting Figure 4.7.

Lemma 4.4.4. *Let $|I_{\text{A}}|$ be the number of available aircraft, F the set of flights ordered by departure time and F^j the set of flights arriving at airport j ordered by departure time. Let $\{f_{j_1}, f_{j_2}, ..., f_{j_{|I_{\text{A}}|}} \subseteq F^j\}$ denote $|I_{\text{A}}|$ subsequent flights, t_i^{Arr}*

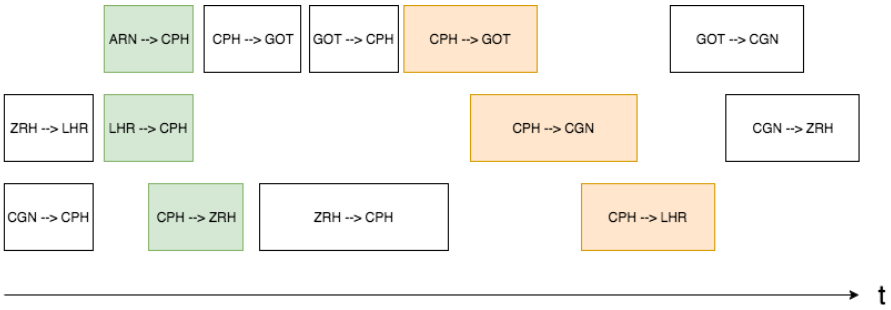


Figure 4.6: An example gantt schedule of flights. The orange marked flights all depart from CPH and no route can be made which cover any two of these flights. Thereby, for a fleet of three aircraft, we can use Lemma 4.4.3 and remove every arc from flights departing before these flights to flights departing after these flights in the corresponding network. During the time period with the green marked flights, all three aircraft need to be utilized simultaneously, and by using Lemma 4.4.2 we may remove each arc from a flight departing before these flights to a flight departing after these flights.

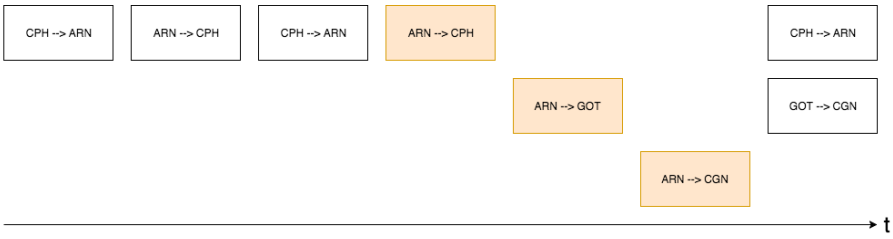


Figure 4.7: An example gantt schedule of flights. The orange marked flights all depart from ARN, and no route can be made which cover any two of these flights. Lemma 4.4.3 can thereby be applied to remove every arc from flights departing before these flights, to flights departing after these flights (if we have a fleet consisting of three aircraft). There is however no subset of flights in this gantt where all aircraft must arrive at the same airport, and Lemma 4.4.4 can not be applied to remove any arcs.

the arrival time of flight i and t_i^{Dep} the departure time of flight i. Let $f_l \in F$ and $f_m \in F$ be two flights such that $t_l^{\text{Arr}} < t_{i_1}^{\text{Dep}}$ and $t_m^{\text{Dep}} > t_{i_{|I_A|}}^{\text{Arr}}$. If no path can be made in the network containing two or more of the flights in the set $\{f_{j_1}, f_{j_2}, ..., f_{j_{|I_A|}}\}$, then no feasible solution to the tail assignment problem can contain a path using an arc between f_l and f_m.

Proof. Using an arc between flight nodes f_l and f_m would mean that one of the flights in the set $\{f_{j_1}, f_{j_2}, ..., f_{j_{|I_A|}}\}$ would be left unassigned. □

Lastly, another way to reduce the number of arcs in the network is to simply set an upper limit on the number of predecessors to every node. That is, for every node in the network, if the node has more than m predecessors, remove the ingoing arcs from the flight with the departure time furthest back in time, until only m predecessors remain. This strategy can however remove the optimal solution from the solution space if, in the optimal solution, one (or several) aircraft stay on the ground for an

extended period of time. for example, if maintenance has to be performed on any of the available aircraft during the time we are planning for, this strategy should not be used. This, since it is probable that we can not make room for a maintenance check if no routes can be generated where the aircraft stands still on the ground for extensive periods of time. Further, if there is reason to suspect that a good feasible solution to the problem will have an uneven aircraft utilization, this strategy should not be used. This, since removing 'far away' predecessors to nodes will benefit routes containing a lot of flights.

There are, however, instances for which this strategy is useful and will significantly reduce the time it takes to solve the tail assignment problem. For example, an airline might have all maintenance checks on weekends, when every aircraft is on the ground at the 'base'. If this is the case, removing distant predecessors will not damage the possibility to make room for maintenance checks in the schedule.

5
Tests and results

The tests were performed on Windows computers with Intel(R) Core(TM) i7-5600U CPU 2.60GHz processors and 16 GB RAM. The mathematical models and algorithms have been implemented in the programming language C# [1], and linear programming problems have been solved using the solver GLOP [2].

5.1 Tools to interpret the results

As we have two different column generation methods and two different solution space restricting heuristics we need tools to compare them with each other. The four main tools we have used are the objective function, computation time, the total number of arcs in the network and the number of efficient labels in the label pulling algorithm.

The number of arcs in the network are initially counted before the algorithm starts, thereafter this number is updated each time the heuristics are applied to the problem. This way we see the relative aggressiveness of the heuristics compared to each other as well as if there is an impact on how fast the solution space is shrinking, depending on what column generation algorithm is used. The latter can also be seen when looking at the number of efficient labels in each column generation iteration.

5.2 Test instances

All test instances are data taken from commercial airlines in the flight industry. We have used data from three different airlines, ASH, VIVA and WOW. The ASH-instance spans over five days and the Viva-instance spans over a week. WOW3d is a schedule spanning over three days, WOW1w spans over one week, WOW2w spans over two weeks, WOW1m spans over a month and WOW2m spans over two months. WOWm1w is the WOW1w test instance of one week with preassigned maintenance. Three of the test instances include flights which make it impossible to create a schedule without unassigned flights, also referred to as ghosts. These are ASH, WOW1m and WOW2m. We include these instances in order to test the robustness of the heuristics and column generation methods.

5.2.1 Objective function

The objective function used for all the results in this thesis is a simple linear function whose value solely depends on the ground time between flights. Denoting the ground

time between flights v_1 and v_2 as $t(v_1, v_2)$, and the minimum turn around time between these flights as $t_{\min}(v_1, v_2)$, the objective function used looks as follows:

$$f(v_1, v_2) = t(v_1, v_2) - t_{\min}(v_1, v_2). \tag{5.1}$$

The lowest cost of an arc using this objective function is 0, which happens if the ground time between the two flights is equal to the minimum turn around time between the two flights.

5.3 The tests

To evaluate the different column generation methods and fixing heuristics, we have tested combinations of them for all the different instances mentioned in Section 5.2. We have for each test instance made runs with all four combinations of standard column generation (Section 3.2) or integer quality column generation (Section 3.3), and the connection fixing heuristic (Section 4.2.2) or the aircraft fixing heuristic (Section 4.2.3). For every test instance where a solution without unassigned flights was obtainable, we have applied the techniques described in Lemmata (4.4.1), (4.4.2), (4.4.3) and (4.4.4) as preprocessing of the flight network.

The point of using several, different sized test instances with data from WOW, is to see how the algorithms and fixing heuristics performance depends on the size of the problem. We can get a good view of this if the we have several test instances with similar underlying flight patterns, as is the case with the WOW test instances. The WOWm1w is used to evaluate how the algorithms can handle preassigned activities. The ASH and VIVA datasets contains more flights and aircraft than the WOW dataset, and are used to evaluate how the algorithms handle larger data and different structures of the underlying flight networks.

For each test run we have kept track of the total number of iterations the algorithms performed before terminating, the average number of columns in the restricted master problem ((3.16) or (3.20)) over the entire run, the objective value of the final solution (using objective function (5.1)), the number of unassigned activities in the solution (also referred to as 'ghosts'), the average number of labels when solving the subproblem (see Section 4.1.1), the total running time of the algorithm, the initial number of arcs in the flight network, and the objective values percentage of the calculated lower bound (Section 4.3). To evaluate the preprocessing methods, we have recorded (for the appropriate test instances) the number of arcs removed when applying them one after another. The results are presented further below, in Tables 5.1 and 5.2.

5.4 Results

5.4.1 Comparing column generation methods

When using IQ-column generation, we can only add one column to the solution space each time we solve the subproblem. For this reason, when using the IQ-method, it is pointless to allow for many labels on each node if no complicating maintenance

constraints are present. This leads to quicker solution times for the subproblem when using IQ-column generation compared to the standard column generation method. Using standard column generation we can add multiple columns to the solution space each time we solve the subproblem. Regardless of the longer solution times in each iteration, the standard column generation approach generally leads to shorter overall running times since fewer iterations are required in order to find a good feasible solution. The number of efficient labels in a typical test run for IQ and standard column generation can be seen in Figure 5.1.

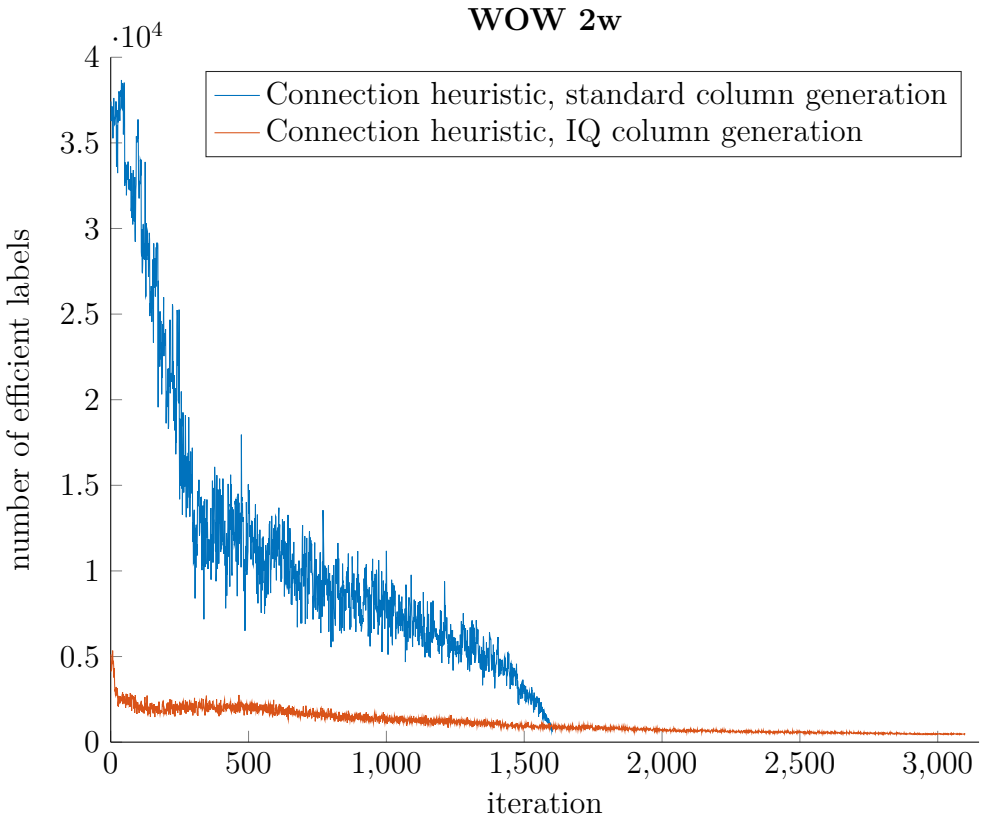


Figure 5.1: Number of efficient labels after each column generation method for the two column generation methods on test instance WOW2w.

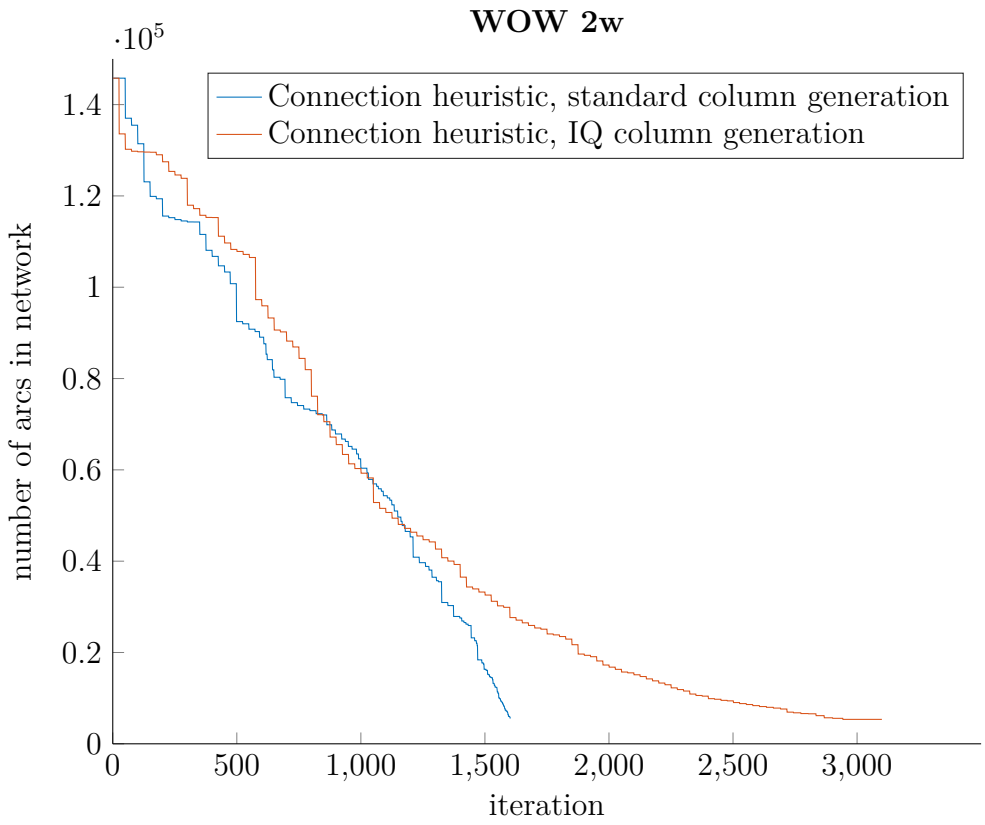


Figure 5.2: Number of arcs in the network after each column generation iteration for the two different column generation methods on test instance WOW2w.

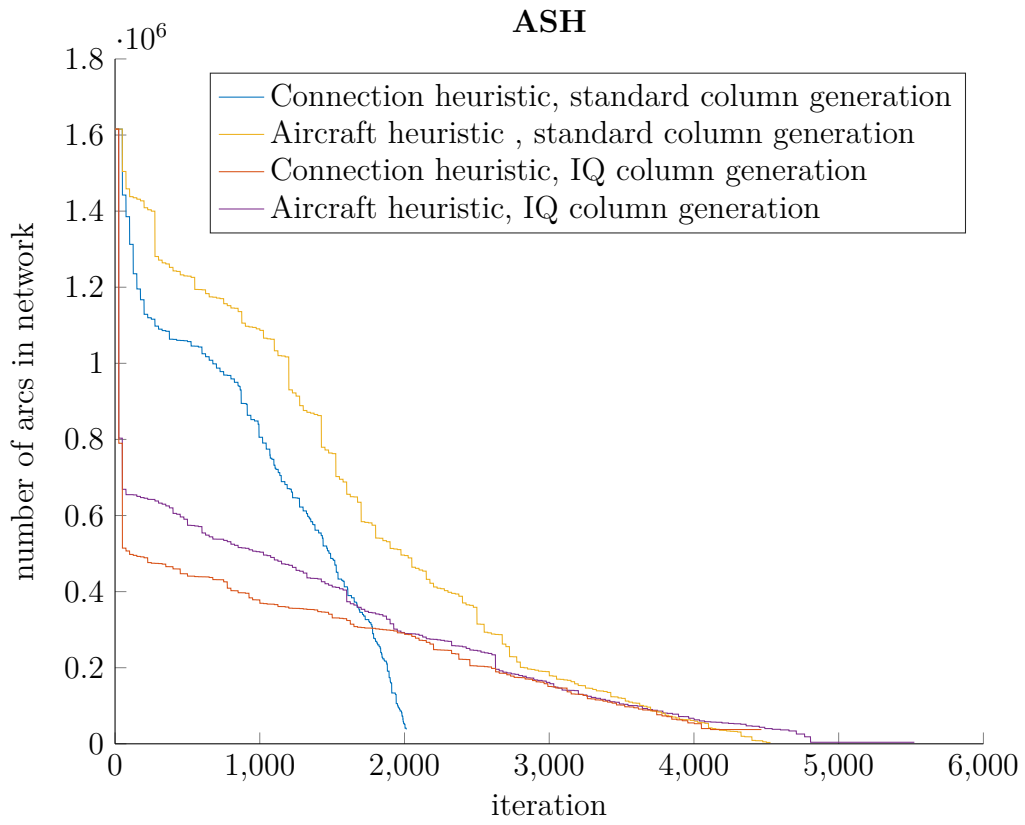


Figure 5.3: Number of arcs in the network for each iteration in the test instance ASH.

In the test instance ASH, IQ-column generation was able to roughly halve the number of arcs in the network to the subproblem after the first time the connection fixing heuristic was applied (see Figure 5.3). Additionally, IQ-column generation was able to halve the number of arcs in the network for test instance VIVA in the first few fixing iterations. These observations has led us to the idea that it might be preferable to combine IQ-column generation with the standard column generation method, using IQ-column generation only in the first few iterations to quickly reduce the size of the solution space, and then switching to the standard column generation method for the remainder of the run. The reader can see that this is not always the behaviour when looking at Figure 5.2, which depicts a test instance in which the IQ and standard column generation algorithms remove arcs in a similar pace. The approach of using both IQ and standard column generation is not treated further in this report.

Regarding the smaller test instances (WOW3d and WOW1w), both IQ and standard column generation perform well. Both methods (regardless of the fixing heuristic used) provides solutions without unassigned flights, and with similar objective values.

Looking at the larger test instances (ASH, VIVA, WOW1m, WOW2m) however, the results vary. On the VIVA test instance, the only method which provided a solution with no unassigned flights was IQ-column generation combined with the connection fixing heuristic. The standard column generation method with the con-

nection fixing heuristic had a shorter running time on this test instance, but resulted in a solution with two unassigned flights, and is thereby considered inferior to the IQ-column generation approach in this case.

In the ASH test instance there exists no solution with all flights covered (which can be seen by solving the minimum cost network flow problem (4.6)), and that none of our methods is able to find such a solution is thereby the only possible result. On this instance, the standard column generation approach performed best, providing a solution with thirteen unassigned flights, compared to sixteen for IQ column generation.

5.4.2 Comparing integer heuristics

Generally the connection fixing outperforms aircraft legality fixing in terms of objective value. However, taking the execution times into account there are a few test cases that suggest that aircraft legality might be a preferred heuristic. For example on test instance WOW2w, aircraft legality fixing together with IQ column generation is twice as fast as the method yielding the lowest objective value, while only being 0.27% more expensive objective value wise (see Table 5.1).

When using IQ column generation, the aircraft legality fixing has shorter running times for every test instance in the WOW database, except for the test instance WOWm1w. The runs with the aircraft legality fixing heuristic also resulted in comparable objective values for these test instances (performing a lot better on the WOW1m instance, and slightly worse on the other ones). On the ASH and VIVA test instances, the aircraft legality fixing heuristic performed worse than the connection fixing heuristic, both in terms of computing time and objective value. This was the case regardless of which column generation method was used.

5.4.3 Results from pre-processing methods

Table 5.2 shows the initial number of arcs for the different test instances, and the number of arcs after the pre-processing methods derived from Lemma 4.4.2 through 4.4.4 were applied. As stated in Section 4.4 these methods are not reliable when the optimal solution to the tail assignment problem contains unassigned flights. For this reason we have not performed any preprocessing on the test instances ASH, WOW1m and WOW2m, and no preprocessing results for these instances are presented in Table 5.2.

The pre-processing methods were applied in the order: Lemma 4.4.2, Lemma 4.4.3, Lemma 4.4.4. This means that some arcs that would have been removed using the method derived from Lemma 4.4.4 might have already been removed using one of the prior methods.

Clearly the methods derived from Lemma 4.4.3 and Lemma 4.4.4 work best on our test instances as they are the only ones that actually reduced the number of arcs. In test instances WOW3d, WOW1w, WOW2w and WOWm1w the airline has a hub to which aircraft often return, and we assume that this is why these methods performed well. We see also that 4.4.2 did not remove any arcs on any test instance, see Figure 5.4.

test instance	column generation method	fixing heuristic	#iter	average #col	objective value	#gho	average #labels	total run time [s]	initial #arcs	% of lower bound
WOW3d	stand	connection	109	323.0	9.12e+03	0	1.15e+03	1.8	1.27e+04	100.0
WOW3d	stand	legality	317	483.0	9.34e+03	0	8.89e+02	5.9	1.27e+04	102.4
WOW3d	IQ	connection	714	203.7	9.13e+03	0	2.23e+02	9.3	1.27e+04	100.1
WOW3d	IQ	legality	502	175.4	9.20e+03	0	2.70e+02	5.0	1.27e+04	100.9
WOW1w	stand	connection	426	676.6	3.24e+03	0	4.05e+03	32.8	4.15e+04	100.0
WOW1w	stand	legality	1329	1174.1	3.33e+03	0	4.19e+03	165.1	4.15e+04	103.0
WOW1w	IQ	connection	1776	350.1	3.24e+03	0	5.02e+02	67.6	4.15e+04	100.1
WOW1w	IQ	legality	708	328.0	3.44e+03	0	6.23e+02	23.9	4.15e+04	106.3
WOW2w	stand	connection	1603	1282.1	7.80e+04	0	1.14e+04	535.6	1.46e+05	100.0
WOW2w	stand	legality	290	573.6	1.85e+05	1	1.92e+04	61.7	1.46e+05	237.6
WOW2w	IQ	connection	3100	566.1	7.81e+04	0	1.14e+03	325.3	1.46e+05	100.1
WOW2w	IQ	legality	2614	558.6	7.82e+04	0	1.26e+03	262.3	1.46e+05	100.3
WOWm1w	stand	connection	273	1545.8	3.20e+03	0	2.85e+03	26.2	3.96e+04	108.1
WOWm1w	stand	legality	1120	3257.1	3.70e+03	0	3.68e+03	271.7	3.96e+04	125.0
WOWm1w	IQ	connection	2166	367.0	2.98e+03	0	1.49e+03	148.5	3.96e+04	100.5
WOWm1w	IQ	legality	2015	360.0	3.21e+03	0	1.97e+03	153.0	3.96e+04	108.6
WOW1m	stand	connection	121	1011.4	2.98e+05	1	1.08e+05	137.7	9.38e+05	106.8
WOW1m	stand	legality	127	1005.4	2.98e+05	1	1.49e+05	258.9	9.38e+05	106.8
WOW1m	IQ	connection	7608	1078.4	1.90e+06	18	5.46e+03	3324.3	9.38e+05	683.0
WOW1m	IQ	legality	3290	1065.9	5.04e+05	4	9.19e+03	2490.8	9.38e+05	180.9
WOW2m	stand	connection	312	1978.2	1.24e+06	8	5.44e+05	2003.0	5.19e+06	159.3
WOW2m	stand	legality	286	1942.3	1.24e+06	8	6.92e+05	2259.0	5.19e+06	159.3
ASH	stand	connection	2015	2526.8	1.39e+06	13	2.43e+04	1775.0	1.62e+06	118.3
ASH	stand	legality	4529	3197.7	2.07e+06	20	3.05e+04	7429.5	1.62e+06	176.2
ASH	IQ	connection	4466	1264.2	1.60e+06	16	2.00e+03	1924.0	1.62e+06	136.3
ASH	IQ	legality	5523	1258.1	2.07e+06	20	3.40e+03	3068.9	1.62e+06	176.4
VIVA	stand	connection	3153	2811.2	2.78e+05	2	3.21e+04	3231.9	9.40e+05	372.6
VIVA	stand	legality	15175	16020.0	3.40e+06	33	4.17e+04	46025.1	9.40e+05	4560.6
VIVA	IQ	connection	6445	1309.1	8.01e+04	0	1.87e+04	4631.7	9.40e+05	107.6
VIVA	IQ	legality	11389	1610.0	2.10e+06	20	7.49e+04	13111.0	9.40e+05	2815.1

Table 5.1: Table of results from test runs. The stand column generation method is the standard one while IQ is s the integer quality column generation method. #iter is the number of iterations before the algorithm terminated. average #col is the average number of columns in the restricted master problem over the entire test run. objective value is the objective value of the final solution of the optimization problem given by the test instance. #gho is the number of unassigned flights in the final solution. average #labels is the average number of efficient labels in the subproblem over the entire test run, i.e. the sum over the total number of efficient labels in each column generation iteration divided by the total number of iterations. total run time [s] is the total run time for the entire algorithm in seconds. The % of lower bound indicates how close the final solution is to the calculated lower bound.

instance	initial #arcs	#arcs 4.4.2	#arcs 4.4.3	#arcs 4.4.4
WOW3d	14676	14676	12708	12696
WOW1w	78264	78264	54369	41496
WOW2w	292956	292956	232968	145788
WOWm1w	79576	79576	79576	43913
VIVA	940495	940495	940495	940495

Table 5.2: The number of arcs in the network after the pre-processing methods derived from Lemmata 4.4.2 through 4.4.4 was applied. The pre-processing methods were applied in the following order: Lemma 4.4.2, Lemma 4.4.3, Lemma 4.4.4.

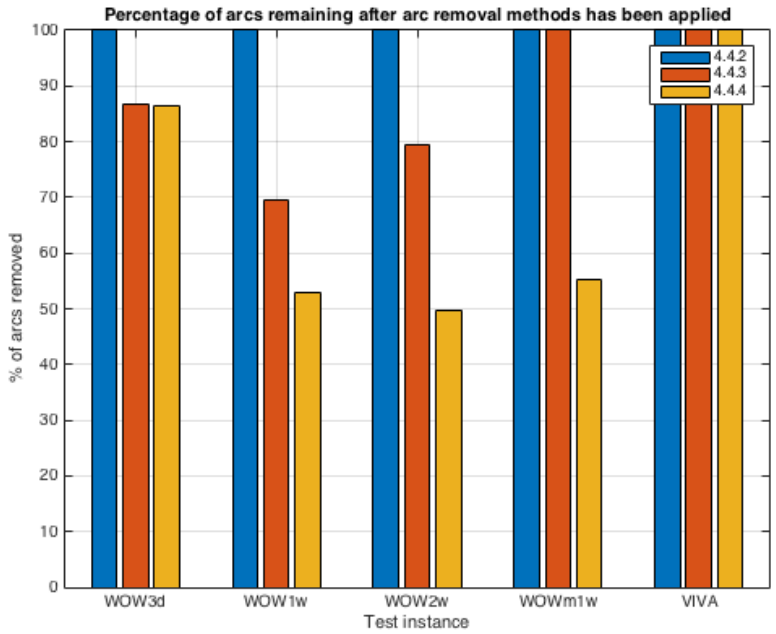


Figure 5.4: Bar plot showing the percentage of arcs remaining after each arc removal method has been applied. It is apparent that Lemma 4.4.2 fails to remove any arcs on any test instance.

We find the most striking result on the WOW2w test instance, where more than half of the arcs in the original network could be removed using the methods derived from Lemmata 4.4.3 and 4.4.4.

On the test instance VIVA, we were not able to remove any arcs. The airline this data set comes from does not have a central hub from where a lot of flights depart and arrive, which is likely the reason we could not remove any arcs by applying Lemmata 4.4.3 and 4.4.4 on this test instance.

6

Conclusion

We have adressed the problem of solving the Tail Assignment Optimization problem formulating the problem as a set partitioning problem and solving this using column generation. Two different column generation approaches have been used, the common column generation method and the IQ column generation. To enforce integer solutions we have used two different integer heuristics, connection fixing heuristic and aircraft legality fixing heuristic. To evaluate the different methods we compare their performance compared to each other by combining them in various ways. The tests have been performed on several test instances taken from real airline data.

Standard column generation together with either connection fixing or aircraft legality fixing has generally performed the best. Even though there are some instances for which IQ column generation has yielded a lower objective function value (i.e. for WOWm1w and VIVA) or lower running time (for WOW1w), on average it produces a schedule in longer time and with a higher final objective value.

The connection fixing heuristic was overall slightly faster and produced lower objective values compared to aircraft legality fixing. The idea behind aircraft fixing is that it should restrict the solution space based on how aircraft specifics affect the columns generated more efficiently than the connection fixing. The data we obtained had less such information than we hoped and therefore the aircraft fixing heuristic did not perform as well as we hoped.

The general strength of standard column generation compared with IQ is our ability to add several columns to the restricted master problem in each iteration. This leads to overall shorter computing times for the standard column generation method when compared to IQ.

In our tests, IQ column generation has performed worse on the instances which lacked a solution with no unassigned flights. An explanation could be that in order to include yet unassigned flights into a path, the column generator needs to create paths including parts of already created paths in order to include an unassigned flight. As a result of this, the multipliers upgraded by the subgradient method could be skewed, making it harder for the column generator to produce columns that fit well with the already created paths.

The algorithms and methods presented in this thesis will be used by Aviolinx as a part of their airline management system RAIDO.

6.1 Future research

As IQ had problems with time complexity due to it only being able to add one column each iteration (see section 3.3) one could possibly change how columns are added by calculating the reduced cost of each generated path as if they were the optimal, namely calculating $(c_j - \sum_{i \in I} a_{ij}\gamma_i)/(\sum_{i \in I} a_{ij})$ and add the n best columns, for some well chosen value of n. The reason why Bredström et al. did not do this was that they want to better control the over coverage. However, it might be possible to tune the algorithm to limit the over coverage and achieve a faster algorithm.

Aircraft legality fixing will probably perform better with data containing more aircraft specifics that can be utilized for the legality fixing. Moreover, one can combine aircraft legality fixing with connection fixing, in order to fix connections that are aircraft specific. This would require a restructuring of the node network, and might make the problem more complex, but could improve the performance of this integer heuristic.

We have implemented pre-processing methods for removing unnecessary arcs from the network by looking at time intervals where all aircraft must be used simultaneously, where all aircraft must arrive at the same airport, and where all aircraft must depart from the same airport. There is, however, a general approach which one could use to remove unnecessary arcs in this manner. The problem of finding a set of nodes where every aircraft must be used can be formulated as a problem of finding a complete subgraph of size $|T|$, where $|T|$ is the number of aircraft. An algorithm which can find a complete subgraph of size k for a graph with n nodes was developed by Vassilevska [22] in 2008. This algorithm runs in $\mathcal{O}(n^k/(\epsilon \log n)^{(k-1)})$ time, $\epsilon > 0$. Both n and k are usually large in our context, so in spite of the fact that the problem of finding a complete subgraph can be solved in polynomial time, we have as of now not looked any further into this approach. If many arcs can be removed in this manner, it might however be worth while. Other solution methods for the problem of reducing the number of arcs on special cases of graphs have later been developed (see for example [18]), but these also have a large polynomial time complexity.

As mentioned in Section 4.1.1, a new method for solving resource constrained shortest path problems is presented in [11]. Implementing this technique to make it work for solving the subproblem for the tail assignment problem could be preferable to the label setting algorithm used within this thesis.

Our last suggestion is the one mentioned in the result section, that of combining the two column generation methods. To apply IQ column generation between the first integer fixing applications and thereafter continue with the standard column generation. This could be beneficial as we have seen that when using IQ column generation the algorithm sometimes restricts the solution space significantly in the beginning of the computations while the standard column generation generally helps to restrict the solution space more effectively in later iterations.

Bibliography

[1] C#. https://docs.microsoft.com/en-us/dotnet/csharp/csharp. Accessed: 2017-07-25.

[2] The glop linear solver. https://developers.google.com/optimization/lp/glop. Accessed: 2017-07-25.

[3] Raido. http://aviolinx.com/product/raido. Accessed: 2017-06-06.

[4] Ravindra K. Ahuja, Thomas L. Magnanti, and James B. Orlin. *Network Flows: Theory, Algorithms and Applications*. Prentice Hall, Upper Saddle River, New Jersey, Englewood Cliffs, N.J., 1993.

[5] Ralf Borndörfer, Ivan Dovica, Ivo Nowak, and Thomas Schickinger. Robust tail assignment. Technical Report 10-08, ZIB, Takustr.7, 14195 Berlin, 2010.

[6] D. Bredström, K. Jörnsten, M. Rönnqvist, and M. Bouchard. Searching for optimal integer solutions to set partitioning problems using column generation. *International Transactions in Operational Research*, 21(2):177–197, March 2014.

[7] G. Desaulniers, J. Desrosiers, and M.M. Solomon. *Column Generation*. Springer, Spring Street, New York, 2005.

[8] Ivan Dovica. *Robust tail assignment*. PhD thesis, Technischen Universität Berlin, 2014.

[9] Gary Froyland, Stephen J. Maher, and Cheng-Lung Wu. The recoverable robust tail assignment problem. *Transportation Science*, 48(3):351–372, 2014.

[10] Sami Gabteni and Mattias Grönkvist. Combining column generation and constraint programming to solve the tail assignment problem. *Annals of Operations Research*, 171(1):61, 2008.

[11] R. Garcia. *Resource constrained shortest paths and extensions*. PhD thesis, School of Industrial and Systems Engineering, Georgia Institute of Technology, 2009.

[12] Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, USA, 1979.

[13] M. Grönkvist. *The Tail Assignment Problem*. PhD thesis, Chalmers University of Technology and Göteborg University, 2005.

[14] Mattias Grönkvist and Jens Kjerrström. Tail assignment in practice. In H Fleuren, D den Hertog, and P Kort, editors, *Operations Research Proceedings 2004*, pages 166–173. Springer, Berlin, 2005.

[15] Marcial Lapp and Florian Wikenhauser. Incorporating aircraft efficiency measures into the tail assignment problem. *Journal of Air Transport Management*, 19:25–30, 2012.

[16] E. Q. V. Martins and J. L. E. dos Santos. The labelling algorithm for the multiobjective shortest path problem. Technical report, Departamento de Matemática, Universidade de Coimbra, Coimbra, Portugal, November 1999.

[17] M. Patriksson N. Andréasson, A. Evgrafov. *An Introduction to Continuous Optimization*. Studentlitteratur AB, Göteborg, 2013.

[18] Benjamin Rossman. The monotone complexity of k-clique on random graphs. *SIAM Journal on Computing*, 43(1):256–279, 2014.

[19] Sebastian Ruther, Natashia Boland, Faramroze G. Engineer, and Ian Evans. Integrated aircraft routing, crew pairing, and tail assignment: Branch-and-price with many pricing problems. *Transportation Science*, 51(1):177–195, 2017.

[20] Elina Rönnberg and Torbjörn Larsson. All-integer column generation for set partitioning: Basic principles and extensions. *European Journal of Operational Research*, 233(3):529–538, 2014.

[21] J. A. Tomlin. Minimum-cost multicommodity network flows. *Operations Research*, 14(1):45–51, 1966.

[22] Virginia Vassilevska. Efficient algorithms for clique problems. *Information Processing Letters*, 109(4):254–257, 2009.

[23] Laurence A Wolsey. *Integer Programming*. Wiley, New York, NY, USA, 1998.