Crew Scheduling

Approach to Calculate the Crew Productivity of Flight Schedules

Maarten Vonk

TU Delft

Crew Scheduling Approach to Calculate the Crew Productivity of Flight Schedules

by

Maarten Vonk

to obtain the degree Master of Science in Applied Mathematics
at the Delft University of Technology,
to be defended publicly on Friday January 18, 2019 at 15:00.

Student number:	4619536	
Project duration:	March 18, 2018 – January 18, 2019	
Thesis committee:	Dr. D. C. Gijswijt,	TU Delft, supervisor
	Dr.Ir J. H. Weber,	TU Delft
	Dr. J. Rezaei,	TU Delft

This thesis is confidential and cannot be made public until January 18, 2019.
Please print in color, if printing this report.

An electronic version of this thesis is available at `http://repository.tudelft.nl/`.

Abstract

With cockpit crew costs being the second largest costs of an airline, making optimal use of the available crew is very important. The productivity of the crew is limited by the labor agreement and law regulations, which prevent the crew members from working irregularly or excessively. In this thesis we present several methods to solve the crew scheduling problem as to calculate the crew productivity based on the labor agreements and law regulations.

The crew scheduling problem is decomposed into the crew pairing problem and the crew rostering problem. A set covering approach is used to solve the traditional crew pairing problem and a matching algorithm is used to solve the crew pairing problem that arises when we allow flights being retimed. The crew rostering problem is tackled by a minimum cost flow network method and a column generation approach. All the methods are tested on a variety of flight schedules deviating in the number of night flights included.

Title: Crew Scheduling Approach to Calculate the Crew Productivity of Flight Schedules
Author: Maarten Vonk,
Date: December 5, 2018

Faculty of Electrical Engineering, Mathematics and Computer Science
TU Delft
Mekelweg 4, 2628 CD Delft

Preface

This thesis has been written as concluding part of the master Applied Mathematics at Delft University of Technology and the work has been carried out at the airline Transavia. I would like to thank my university thesis supervisor Dion Gijswijt for all his help with the mathematical concepts and structure of the thesis, Wouter Stikvoort and Karin van Eeden at Transavia for their guidance at a daily basis and my team integral planning for making it possible for me to carry out this thesis.

Maarten Vonk
Amsterdam, December 2018

Contents

1. Introduction

In addition to fuel costs, crew costs are one of the highest expenses of an airline. Crew consists of cabin crew and cockpit crew, but the costs for cockpit crew members are exceedingly larger than the costs for cabin crew members. As any other employee, pilots and co-pilots receive a monthly salary, but their working days and hours can vary heavily depending on the composition of the flight schedule. Beside the fact that a pilot cannot perform multiple flights at the same time, restrictions on rostering crew members consist of the location of the crew member at a given time and many labor agreements and law regulations that protect the pilots from working excessively or irregularly. It is of paramount importance that the pilots are fit during their duties. Therefore, the labor agreements are a bit stricter when they concern flights during the night time hours which may lead to a reduction of productivity of crew members who execute a lot of these night flights. All these rules and regulations constitute the *crew scheduling problem*, the assignment of individual crew members to flights such that the rosters conform with labor and law regulations.

1.1. Literature Overview

Traditionally, airlines divide the crew scheduling problem into two separate problems, the *crew pairing problem* and the *crew rostering problem* [1]. While the goal of the crew pairing problem is to string the flight duties together to create *pairings*, unassigned work packages that start and end at the same location, the crew rostering problem concerns the assignment of these pairings to individual crew members. These problems are solved separately because of the high complexity of the crew scheduling problem. However, because solving these problems sequentially may lead to suboptimal solutions, some researchers have tackled the crew rostering problem in its entirety [8, 20].

The objective of the crew scheduling problem can vary. Mostly, the crew scheduling

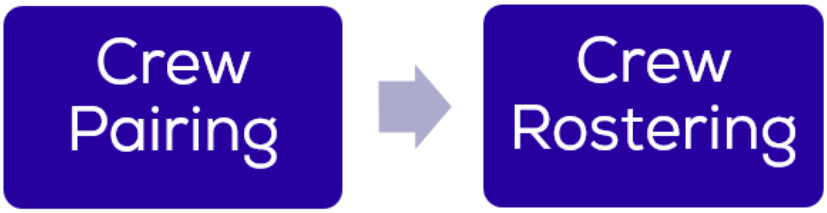

Figure 1.1: Crew Scheduling

problem is solved when the number of available crew members has already been identified. The goal of the crew scheduling is then to balance the workload, to optimize the robustness of the schedule or to account for individual preferences of crew members [9, 11, 12]. But in order to have an idea of how many crew members should be hired, another optimal crew scheduling problem should be solved previously to calculate the minimum number of crew members necessary to perform a certain flight schedule. Because this crew scheduling problem does not assign individual crew members to pairings, but determines the minimum number of crew members necessary to realize a schedule, sometimes this problem is treated as an extended pairing problem [6]. The benefit of solving the optimal crew scheduling problem long in advance is that the flight schedule can be adjusted accordingly. The most common formulation for the crew pairing problems as well as the crew rostering problems is the set covering formulation [9, 11, 19, 15, 2, 20]. Due to the enormous number of variables considered, many scholars use a column generation approach to solve the set covering problem [2, 9, 11, 20]. A more detailed overview of the various methods used by scholars to solve the crew scheduling problem is given in Chapter 4.

1.2. Summary of Methods and Results

This thesis focuses mainly on optimal crew scheduling. Given various flight schedules, we try to solve the crew scheduling problem for optimality of certain key performance indicators such as the minimum number of crew members required. Since we will show that the flight schedules we are dealing with already imply the construction of the majority of the pairings, we focus mainly on the crew rostering problem. However, we do include some solution methods to solve some of the pairing problems that are not previously determined and as an extension, a graph matching method is used to compute optimal pairings when we allow flight retimings. We exploit several mathematical methods to solve the crew rostering problem such as a layered graph approach based on a minimum cost flow network [16] to solve the minimum number of crew members necessary and the minimum amount of idle time as well as a column generation approach for crew scheduling [2]. Additionally, we created a bottleneck model that optimizes the schedule for *block time*, the time the aircraft is in the air, and identifies the pairings that are the least productive.

To compare the various mathematical approaches, we test their accuracy performance as well as their computational performance on a variety of flight schedules based on the schedule of the summer period of 2019 consisting of 677 flights per week. We solve the daily pairing problem according to traditional method and solve it when we allow flights being retimed and a slight improvement of retimed pairings is observed. While the column generation approach to optimal crew scheduling allows for a lot of features and has room for many additional constraints, its computation time shows very poor results and even with rapid branching strategies it is not apt to solve our large crew scheduling problem for optimality. The layered graph approach has very promising computational results but the constraints that can be set on the roster-lines individually are very limited.

However, it works well enough to analyze the impact of certain flight schedules for the productivity of the crew. By testing the model on a variety of flight schedules, we see how the reduction of flights during the night leads to increased productivity, but this effect is only limited.

1.3. Research Questions

In order to structure this thesis, we initially formulated two research questions. The first research question encompasses the formal exploration of the problem and gives rise to the construction of a mathematical simulation model.

> *What is the exact impact of various sorts of night flights concerning the additional labor requirements of night flights compared to day flights and how do these requirements affect the productivity of the cockpit crew?*

The second research question relates to the results of the mathematical model and involves the practical relevance of the problem.

> *What are the exact costs we make when utilizing an extra aircraft and exact costs we save on cockpit crew when we use that aircraft to replace night flights with day flights?*

We get back to the initial research questions in Chapter 9, when we summarize the answers to these questions or explain why we have deviated from the initial research questions.

In Chapter 2 and Chapter 3 we give a brief introduction of the mathematical concepts we will use throughout this thesis. In Chapter 4 a thorough description of the problem analysis is presented, including a general overview of the roster process as well as the technical details involved. We propose several methods to improve the crew pairing solution methods compared to the traditional crew pairing solution method in Chapter 5. A motivation of the solution methods for crew rostering is presented in Chapter 6 and we discuss the solution methods for crew rostering in Chapter 7. In Chapter 8 the results we obtained from the crew rostering solutions methods as well as the crew pairing solution methods are discussed. These results are summarized and future research directions are proposed in Chapter 9.

This work has been carried out at the airline Transavia within the team Integrated Planning. The flight schedules and labor regulations used in this paper have been extracted from the airline.

2. Integer Linear Programming

In this chapter we give an introduction to integer linear programming that is relevant for this thesis. The mathematical concepts, notations and terminology will be extensively used throughout this thesis. First we introduce the concept of linear programming. Linear programming is a method for optimizing a given objective function over a subset of $\mathbb{R}^n$ described by m linear inequalities. We write the linear program in the form of the set covering problem, which we frequently use:

$$\min \sum_{i=1}^{n} c_i x_i \tag{1.1}$$

$$\text{s.t.} \quad \sum_{i=1}^{n} b_{i,j} x_i \geq d_j \qquad \forall j \in \{1, \ldots, m\} \tag{1.2}$$

$$x_i \geq 0 \qquad \forall i \in \{1, \ldots, n\}. \tag{1.3}$$

In the given formulation, (1.1) is the *objective function* consisting of the *variables* x_i to be determined and the cost values $c_i \in \mathbb{R}$ for $i \in \{1, \ldots, n\}$. The inequalities represented by (1.2) are the *constraints* with $b_{i,j}, d_j \in \mathbb{R}$ for all $j \in \{1, \ldots, m\}$. Inequalities (1.3) are called the nonnegativity constraints and inequalities (1.2) and (1.3) together shape the *feasible region*, which is the convex set over which the objective function should be determined. An element in the feasible region is a *feasible solution*. For a feasible solution x, an inequality is called *tight* if equality holds for x.

The minimum value of a linear program (LP) is related to the maximum value of a *dual* linear program, which we can formulate relative to the *primal* linear program (1.1)–(1.3) by:

$$\max \sum_{j=1}^{m} d_j y_j \tag{2.1}$$

$$\text{s.t.} \quad \sum_{j=1}^{m} b_{i,j} y_j \leq c_i \qquad \forall i \in \{1, \ldots, n\} \tag{2.2}$$

$$y_j \geq 0 \qquad \forall j \in \{1, \ldots, m\}. \tag{2.3}$$

For a feasible solution x and y for the primal and dual linear program respectively, the relation between the primal and the dual linear program can be formulated by the following system of the inequalities:

$$\sum_{i=1}^{n} c_i x_i \geq \sum_{i=1}^{n} \left(\sum_{j=1}^{m} b_{i,j} y_j \right) x_i \tag{3.1}$$

$$= \sum_{j=1}^{m} y_j \left(\sum_{i=1}^{n} b_{i,j} x_i \right) \tag{3.2}$$

$$\geq \sum_{j=1}^{m} d_j y_j. \tag{3.3}$$

The system of inequalities proves *weak duality*, meaning that the value of a feasible solution in the primal linear program (1.1)–(1.3) is greater than or equal to the value of any feasible solution in the dual linear program (2.1)–(2.3). The *strong duality* theorem implies that the minimum value of the primal linear program (1.1)–(1.3) equals the maximum value of the dual linear program (2.1)–(2.3), given that both are feasible. For a proof of this theorem, we refer to [5]. Another important relation between the primal and the dual program is the *Complementary Slackness* theorem. This theorem gives necessary and sufficient criteria for optimality:

Theorem 2.1 (Complementary Slackness [5])**.** *Let x and y be feasible solutions to the primal and dual LP respectively. Then inequality (3.1) is tight if and only if (4.1) holds and inequality (3.3) is tight if and only if (4.2) holds.*

$$\forall i \in \{1, \ldots, n\} \qquad \textit{either} \quad c_i = \sum_{j=1}^{m} b_{i,j} y_j \qquad \textit{or} \quad x_i = 0 \tag{4.1}$$

$$\textit{and} \quad \forall j \in \{1, \ldots, m\} \qquad \textit{either} \quad d_j = \sum_{i=1}^{n} b_{i,j} x_i \qquad \textit{or} \quad y_i = 0. \tag{4.2}$$

In this thesis we often require the variables to be integer. Therefore, we add integrality constraints $x_i \in \mathbb{Z}$ for all $i \in \{1, \ldots, n\}$ to the system of equations. The resulting problem is called an *integer linear program* (ILP):

$$\min \sum_{i=1}^{n} c_i x_i \tag{5.1}$$

$$\text{s.t.} \quad \sum_{i=1}^{n} b_{i,j} x_i \geq d_j \qquad \forall j \in \{1, \ldots, m\} \tag{5.2}$$

$$x_i \geq 0 \qquad \forall i \in \{1, \ldots, n\} \tag{5.3}$$

$$x_i \in \mathbb{Z} \qquad \forall i \in \{1, \ldots, n\}. \tag{5.4}$$

Sometimes the ILP can only take variables that are 0 or 1. In this situation constraint (5.4) can be further specified to $x_i \in \{0, 1\}$ and constraint (5.3) can be removed. The

problem is then called a *0-1 ILP*. While linear programs are solvable relatively easy (in polynomial time), integer linear programs are much harder to solve (NP hard). In this thesis we use the state of the art ILP-solver Gurobi to solve general integer linear programs. Because we use other techniques to solve 0-1 ILPs throughout this thesis as well, we will also discuss other methods to solve integer linear programs in this chapter. Therefore, the general idea of an algorithm that is widely used to solve integer programs called the branch and bound (B&B) algorithm is discussed. For a formal description of this algorithm we refer to [4].

2.1. Branch and Bound

The general idea of the branch and bound algorithm is to split the feasible region into multiple sets and then explore the feasible solutions in these sets systematically without computing the value of every feasible solution. In the algorithm we make use of the *LP-relaxation*, which is the system of inequalities when we remove the integrality constraints while maintaining the bounds of the original problem. The algorithm evaluates the sets of feasible solutions in a rooted tree structure where the root node contains the complete feasible region. We describe the basic concept of the algorithm based on a minimization problem. At every node an optimal solution to the LP-relaxation is computed and

- If the LP-relaxation on that node is unfeasible, the feasible region is not evaluated any further and the tree is pruned by infeasibility.
- If the LP-relaxation on that node has an optimal solution that is integer, there is no need to split the feasible region any further and the tree is pruned by integrality. The objective value of the solution is an upper bound on the optimal solution.
- If the LP-relaxation on that node has a fractional solution and the objective value of this solution is higher than the lowest upper bound, then there exists no further better solution in the feasible region and the tree is pruned by bound.
- If the LP-relaxation on that node has a fractional solution and the objective value of this solution is lower than the lowest upper bound, then the feasible region is split into two parts where the fractional solution is excluded in both parts.

The efficiency of the algorithm depends heavily on the branching strategies incorporated, which can be chosen specifically for the problem. Most commonly, when x_i is a variable with fractional value $x_i \in (0, 1)$, we split the feasible region by imposing integer value $x_i = 0$ or $x_i = 1$ in the next node. In Chapter 7 a branching strategy is introduced that best fits the problem considered. We use the branch and bound framework embedded in a column generation approach to solve ILPs.

2.2. Column Generation

A well-known method to solve large-scale linear programs is *column generation*. The idea of column generation is that the number of variables in the linear program we want

to solve is too vast to consider all variables so one starts with a small subset of variables and only adds more variables if they have the potential to improve the objective function. The variables not considered are implicitly equal to 0. The problem is decomposed into two problems, the so called *restricted master problem* and the *pricing problem*. While the restricted master problem is a restriction of the original problem with only a small subset of variables considered, the pricing problem is a problem that identifies new variables that can potentially improve the objective function of the *initial master problem*. We keep iterating between the restricted master problem and the pricing problem to add variables that can improve the objective function of the initial problem.

We return to the 0-1 integer program to describe the process and call the relaxation of this problem the initial master problem (MP):

$$\begin{aligned} & \min \sum_{i=1}^{n} c_i x_i & & \\ \text{s.t.} \quad & \sum_{i=1}^{n} b_{i,j} x_i \geq d_j & \forall j \in \{1, \ldots, m\} & \qquad \text{(MP)} \\ & x_i \geq 0 & \forall i \in \{1, \ldots, n\}. & \end{aligned}$$

Because the initial set of variables $\{1, \ldots, n\}$ is too large, only a small subset of variables $I \subset \{1, \ldots, n\}$ is considered, which can produce a feasible solution. The problem arising accordingly is the restricted master problem (RMLP). By solving the RMLP we obtain optimal primal and dual solutions, $\mathbf{x}$ and $\mathbf{y}$ respectively. Using the dual variable $\mathbf{y}$, we want to find a $k \in \{1, \ldots, n\}$ that minimizes the *reduced cost* $\overline{c_k} = c_k - \sum_{j=1}^{n} y_j b_{k,j}$ of a variable. Note that this is the most violated dual constraint when the reduced cost is negative. Because every dual constraint corresponds to a variable of the primal linear program, the reduced cost can be interpreted as the potential improvement in the objective function when we add the variable to the RMLP. This is called the pricing problem. The pricing problem can yield an optimal solution of negative reduced cost. In that case, adding the variable will reduce the objective function of the RMLP and therefore improve the solution of the RMLP.

Then, this process is repeated by solving the RMLP again but now with the augmented set I. When the pricing problem yields an optimal solution of nonnegative reduced cost, the subset I of the RMLP already contains all the variables necessary to produce the same optimal solution as the master problem. In this case one observes the optimal solution of the RMLP and if this is integral, the process ends and the optimal solution to the integer linear program is obtained. Else, we start the branching strategy and we repeat the process in a new node. One can either choose to keep the subset of variables that remain feasible after the branching or one can eliminate them.

3. Minimum Cost Flow Problem

An optimization problem that is widely taken as basic model within this thesis is the *minimum cost flow problem*. In order to give the reader some knowledge about this problem beforehand, we give a brief introduction about this problem. The general idea of the minimum cost flow problem is to send a certain amount of *flow* with minimum costs through a directed graph subject to capacity and flow conservation constraints. In a general minimum cost flow problem, there are multiple *supply* and *sink* nodes, where the flow will start and end respectively. The problems considered throughout this thesis only contain one supply node and one sink node, hence the problem description is limited accordingly in this introduction as well.

We introduce a digraph $G = (V, A)$. The nodes V are connected by arcs $a \in A$ where $A \subseteq V^2$ and the nodes are appended with supply values s_v, which are 0 for all nodes $v \in V \backslash \{s, t\}$. In order to make sure there exists a feasible solution, the sink node t has the negative supply value of the start node s, hence $s_s = -s_t$. See Figure 3.1 for an example of such a digraph. Every arc is appended with cost c_a, minimum capacity $l_a \geq 0$ and maximum capacity u_a. The variable we want to determine is the amount of flow x_a for each $a \in A$. The flow on arc a should at least be as much as the minimum capacity, but cannot exceed the maximum capacity of the corresponding arc:

$$l_a \leq x_a \leq u_a \quad \forall a \in A.$$

For each node $v \in V$, we define the set of arcs going out of the node as $\delta^+(v) = \{a \in A | a = (v, w) \quad \text{for} \quad w \in V\}$ and the set of arcs going into the node as $\delta^-(v) = \{a \in A | a = (w, v) \quad \text{for} \quad w \in V\}$. Evidently, we want that the flow going out a node equals the flow going in that node except for the start and sink nodes. This leads to the *flow conservation constraints*:

$$\sum_{a \in \delta^+(v)} x_a - \sum_{a \in \delta^-(v)} x_a = s_v \quad \forall v \in V.$$

Now we can formulate the minimum cost flow problem in terms of a linear program

$$\begin{aligned} & \min \sum_{a \in A} c_a x_a & & \\ \text{subject to} \quad & x_a \leq u_a & \forall a \in A & \\ & x_a \geq l_a & \forall a \in A & \\ & \sum_{a \in \delta^+(v)} x_a - \sum_{a \in \delta^-(v)} x_a = s_v & \forall v \in V. & \end{aligned} \tag{3.1}$$

There are several algorithms that can solve a minimum cost flow problem. One can

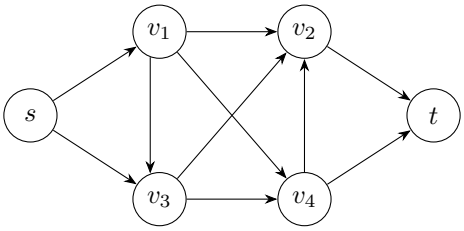

Figure 3.1: Minimum Cost Flow Network

solve the problem by traditional LP solution methods such as interior point method or the simplex method. We can also use the state of the art solver Gurobi. However, this problem's unique structure allows more efficient algorithms to solve this problem such as the network simplex algorithm or capacity scaling algorithm, which are computationally very fast algorithms. In this paper we use the capacity scaling algorithm to solve the minimum cost flow problem.

The reason the minimum cost flow problem is emphasized, is because it attains a beneficial quality based on the following theorem.

Theorem 3.1. *[5] If u, l and s are integer valued and (3.1) has an optimal solution, then (3.1) has an optimal solution that is integral.*

This theorem points out that the solution of the integer program is exactly the same as the solution of the linear program. Therefore, we can use solution methods to solve the linear program in order to obtain the solution of the integer program. Even though in many of the applications throughout the paper, the traditional network flow model is not suitable to describe the entire problem we want to solve, we find out that using its formulation can help solving resembling problems.

3.1. Matchings

Another optimization problem that is discussed in one of the solution methods is the matching problem. Therefore, we briefly discuss this problem and its terminology in this chapter. We introduce an undirected graph $G = (V, E)$. A *matching* M is a subset of edges where no two edges share a common node. A node that is not an endpoint of an edge in the matching M is called an *M-exposed* node. A *maximum matching* is a matching with the minimum number of exposed nodes. A *perfect matching* is a matching in which the edges contain all the nodes.

In addition to this we allow weights w_e on all the edges $e \in E$. The weight of the matching is the sum of the weights of the edges in the considered matching. In this thesis we consider *maximum weight matchings* and *maximum weight maximum cardinality matchings*. In the maximum weight matching problem we optimize the matching for the maximum weight, while in the maximum weight maximum cardinality matchings we optimize for the maximum weight under the constraint that the matching is maximum as well. In Figure 3.2 is an example of two matchings indicated in red. The left-hand

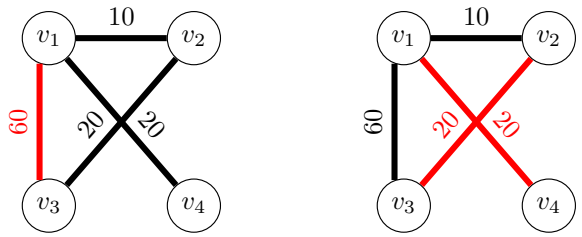

Figure 3.2: Matching examples

matching is a maximum weight matching and the right-hand matching is a maximum weight maximum cardinality matching. In this case, the right-hand matching is also a perfect matching since no node is M-exposed. To solve these matching problems, we use an algorithm based on the blossom method for finding augmenting paths and the primal-dual method for finding a maximum weight matching [5].

4. Problem Analysis

In this chapter we will give a general overview of the planning stages and we specify where this thesis fits within the literature that exists for crew pairing and crew rostering. The problem analysis and the corresponding technical details are also thoroughly described.

4.1. Overview Planning stages

There are four main stages within airline scheduling: the first stage is the flight scheduling problem (FSP) where the *flight schedule* is created that contains the departure times, arrival times, origins and destinations while taking into account passenger demand and ticket price. The second stage is the fleet assignment problem (FAP), where is determined which aircraft type will be used for which flight such that all flights are covered. Then the aircraft maintenance routing problem (AMRP) aims to assign a specific aircraft to specific flights while keeping maintenance feasibility in mind. The last is the crew scheduling problem (CSP), which is separated into two different problems: the crew pairing problem (CPP) and the crew rostering problem (CRP). The crew pairing problem involves the creation of so-called pairings or work packages that can be executed by a random crew member. After the crew pairing, these pairings will then be assigned to specific crew members such that an individual roster is created conform labor en law restrictions. This is called the crew rostering problem.

Traditionally, airlines solve airline schedule planning problems sequentially where the solution of one problem is the input of the consecutive problem (see Figure 4.1). This may result in very poor planning results, because an optimal result of one stage may cause undesired restrictions for the next planning stage. Although it seems efficient to integrate these planning stages, little research has been done so far on solving these planning stages integrally due to the many restrictions and computational complexity of each individual planning stage. The research on each of the planning stages separately as well as the research on some of the planning stages integrally that exists is summarized [7]. Most of the research on the planning stages solved integrally take the crew

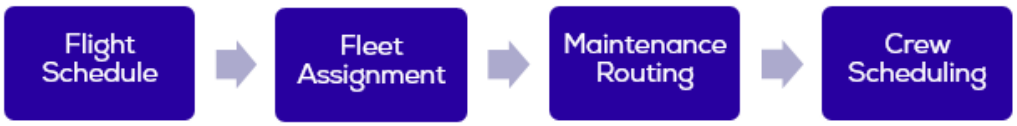

Figure 4.1: Planning Stages

scheduling problem as the objective and involve the fleet assignment and maintenance routing problem as constraints [10, 13]. Others aim to solve the crew scheduling problem integrally, optimizing crew pairing and crew rostering simultaneously [20, 8].

This thesis aims to solve the crew scheduling problem directly after the flight schedule planning assuming fleet assignment and maintenance routing are solved in the flight schedule planning stage. Since this thesis is limited to cockpit crew members, the crew scheduling problem considered only consists of the scheduling of cockpit crew members, which are captains and first officers. Because the pairing problem considered has a very specific character, we decided to solve the pairing problem and the rostering problem separately.

4.1.1. Crew Pairing

The crew pairing problem takes as input the flight legs in the flight schedule that need to be covered. A *leg* is a scheduled flight from take-off to landing. Legs are then grouped to form *duty periods*, which are series of legs together that form a work day for crew. Duty periods are subject to many restrictions such as that the legs of one duty period should follow each other sequentially in time and space. But duty periods should also conform with a minimum *turn around time*, the time between two consecutive legs.

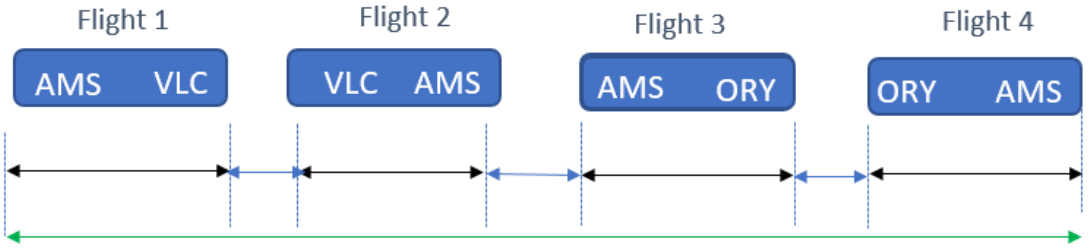

Figure 4.2: Example of a duty period in green and leg in black

Sometimes a duty period ends at a different airport than it starts. In that case the crew can have a *lay-over* until the next duty period begins. Also, a duty period can start at a different base than it ends. In this thesis we consider bases in Amsterdam, Rotterdam and Eindhoven. Any airport that is not a base is called a *foreign airport*. The duty periods are finally converted to *pairings*, which are complete work packages for crew members conform with all labor and law regulations, starting and ending at the same base appended with check-in time before the first departure and check-out time after the last landing.

Figure 4.3 is an example of a pairing that contains four legs. Between the legs are the turn around times, which is the time at an airport where the passengers leave and enter the aircraft and the aircraft needs time depending on the activity of the aircraft (fueling, cargo, etc). Before the departure of the first flight in a duty period is the check-in time and after the last flight of a duty period we have the check-out time. The check-in time

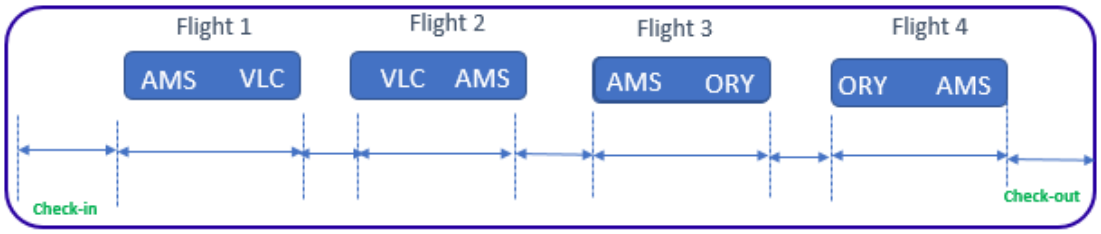

Figure 4.3: Example of a pairing with 4 legs

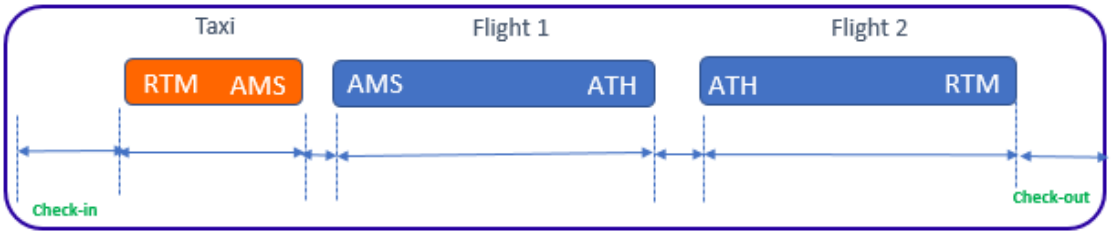

Figure 4.4: Example of a pairing with taxi attached

is exactly one hour before the first departure and the check-out time is exactly half an hour past the last arrival. The complete pairing cannot exceed the maximum amount of working time, the time from check-in to check-out. Note that the pairing in Figure 4.3 can be split into two separate pairings. Duty periods consisting of two legs that can be extended with another duty period of two legs to form a longer duty period and eventually a feasible daily pairing are called *link duty periods* and the associated pairings containing four legs are called *double pairings*. Similarly, pairings that only contain two legs are called *single pairings*. Another example is illustrated by Figure 4.4. In this pairing the duty period starts and ends in a different base, hence the duty period can only be extended to a pairing by including a taxi ride from Rotterdam to Amsterdam. Note that the taxi ride is appended before the duty period in the pairing of Figure 4.4. The taxi ride could have been appended after the duty period as well.

Most of the literature devoted to the crew pairing problem focuses on pairing optimization with respect to pairing costs. However, the flight schedules considered throughout this paper do not allow a lot of room for optimization. This is because the flight schedule consists vastly of outbound legs followed by inbound legs, starting and ending at the same base, which immediately constitutes the pairing. The double pairings such as Figure 4.3 have been pre-assigned by the guidelines in the so-called *crew linking document* [17], a document of pre-assigned pairings that determines which link duty periods are merged to form double pairings as well as which legs are merged to form lay-overs. But there still remains some pairing constructions where the crew linking document does not have strict guidelines for and in many cases the guidelines of the crew linking document are not optimal. In this thesis we use the pairings constructed by the crew linking document as well as our own pairing optimization model. We also include a pairing retiming model

where we allow slight changes in aircraft departure and arrival times to potentially create better pairings. Unlike the crew scheduling models with pairing retiming possibilities of [10] and [13] that use column generation approaches, the unique composition of the legs in the flight schedules considered gives us the opportunity to use a simpler approach of pairing retiming based on graph matching.

4.1.2. Crew Rostering

When all the pairings are created, the next step in the crew scheduling problem is to assign the pairings to individual crew members to create a *roster-line*, a sequence of pairings and/or other duties assigned to one individual crew member conform law-regulations and labor requirements. All the roster-lines together make up the *roster*. We separate two kinds of crew rostering problems.

The first problem is the *balanced* or *gross crew rostering* problem. In this problem we assume to have a fixed number of crew members and the pairings are combined with rest periods, vacations, training duties and weekends so that eventually individual work schedules are generated. The goal of this process can be to equally divide the work load while taking into account individual preferences of crew members, but scholars vary in their goal when studying the gross crew scheduling problem. Some try to solve the crew scheduling problem with the objective to create schedules that have regular work times or contain work patterns [9, 11], while other scholars attach more value to the robustness of the schedule [3].

The second crew rostering problem is the *optimal crew rostering* problem where we allow the number of crew members to vary, but we limit ourselves to rostering the pairings only. The goal is then to minimize certain key performance indicators such as the minimum number of required crew members to perform a schedule, the total idle time of the crew members in the roster or the total number of working hours. This roster process is part of the manpower planning stage, where is determined long in advance how many crew members should be hired and how their hours should be allocated. By doing this process beforehand, we can still allow for minor modifications in the flight schedule if that turns out to improve some of the key performance indicators. Most of the research has been done for gross crew rostering. In this thesis we focus mainly on optimal crew rostering.

A traditional way to state either one of the crew rostering problems is the set covering problem, used by many scholars [9, 11, 19, 15, 2, 20]. Since the set covering problems is NP-hard and the crew rostering problems considered by the scholars differ, the solution methods of these scholars vary. Most of the crew rostering approaches based on the set covering problem contain column generation techniques where some combine column generation with heuristic approaches [2, 9, 11] and others combine pure ILP solution methods with constraint logic programming to create a hybrid column generation approach [20]. Other approaches are based on scatter search algorithms [12], a network flow formulation [16] and a solution method based on a LP-neighborhood structure within tabu-search metaheuristics [19]. In this thesis we use a network flow formulation [16] as well as a column generation approach [2] and a branching strategy accordingly [15].

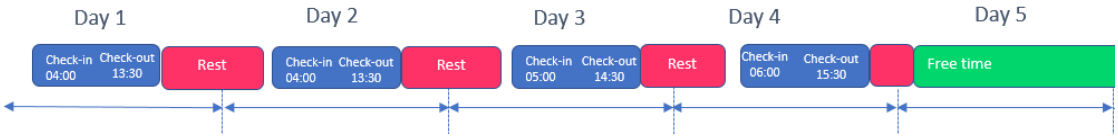

Figure 4.5: Example of a roster-line of 5 days including free time period

Similarly as the crew pairing problem, the crew rostering problem is also bound by limitations imposed by the labor agreements and law regulations. To remedy fatigue of the crew, every pairing is followed up by a mandatory rest period. The length of the rest period depends on the check-out time of the pairing, where is taken into account that working irregular hours requires more rest time to recover. Also, a complete roster period of 35 days contains at most 20 working days. This corresponds with the traditional crew working pattern of maximum 4 days of work and 3 days of free time, which we use throughout this report. The free time period consists of a continuous period of 72 hours which may or may not overlap with the rest period of the last pairing depending on the check-out time of the last pairing. An example of a 5-day roster-line including free time at the end is given by Figure 4.5. Note that in this case the free time period indicated by the green bar overlaps with the rest period of the last pairing. In the next section we specify the details in the labor agreement and law regulations that constitute the guidelines for the crew pairing and crew rostering problem.

4.2. Technical details

As stated before, the crew pairing stage and the crew rostering stage are solved separately in this thesis. Therefore we also separate the labor regulations that concern the crew pairing from the labor regulations that concern crew rostering although they sometimes do overlap. All the specifications of these regulations have been extracted from the labor agreement [18]. Because in all the cases mentioned the labor agreement is stricter than the law requirements [14], the law requirements are omitted in this chapter. Throughout this thesis we focus mainly on the impact of night flights within a flight schedule. In the definition of crew rostering, a *night flight* is a flight for which the pairing in which it is contained has a check-out time between (and not included) 01:00 and 06:00. The corresponding pairing is called a *night pairing*.

4.2.1. Crew Pairing

In the crew pairing stage we discern different labor agreements divided in the maximum work time, the next check-in and check-out times and the minimum required rest time after a pairing. First we describe the maximum work time, which is the time from the check-in to the check-out of a pairing. The length of the maximum work time depends on the check-in time, where is taken into account that crew members can work more when they work regular hours. Therefore, the check-in time has a parabolic relation

with the maximum allowed working time, which is given in Table 4.1

Check-in	04:00-05:45	05:46-08:00	08:00-13:00	13:00-15:45	15:45-19:00
Max work	12:00-13:30	14:00-15:55	16:00	15:55-14:00	13:40-12:00

Table 4.1: Maximum working hours per check-in time frame

The check-in time of a certain pairing also affects the check-in time of a pairing the next day. In order for crew members to keep a certain structure in their time table and to limit the number of changes between early workdays and late workdays, several rules have been incorporated in the labor agreement. In Table 4.2 the limitations on the check-in times given the previous check-in times are presented by the minimum number of hours between the first and the second check-in time. An example of how this pairings regulation works in in practice is given by Figure 4.6.

Check-in	00:00-02:29	02:30-03:59	04:00-05:59	06:00-07:59	$\geq$ 08:00
Next Check-in	27h later	26h later	24h later	23h later	$\geq$ 05:30

Table 4.2: Minimum number of hours between check-in time of two pairings

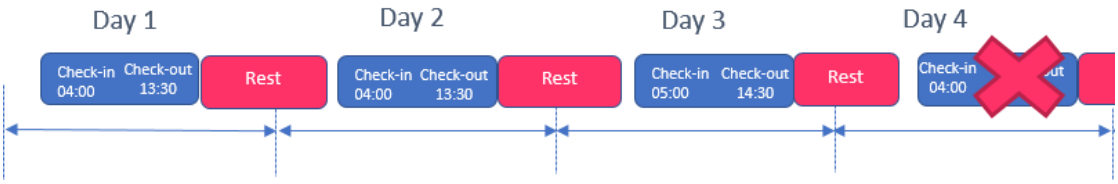

Figure 4.6: Example where pairing at day 4 cannot be sequential to pairing at day 3

Similarly for check-in time, the check-out time of a pairing also affects the next check-in time of a pairing. The restrictions on the next check-in time are first defined by the mandatory rest requirement after a pairing, which also depends on the check-out time. Additionally, there are extra check-in restrictions to prevent early pairings to be sequential to late pairings. In Table 4.3 we describe the minimum rest and the restrictions imposed on the check-in time of the pairing the next day based on the check-out time of the previous pairings (given that the previous pairing starts before or including 23:59).

Check-out	until 02:00	02:01-03:00	03:01-04:00	04:01-05:00	from 05:01
Rest	13h	13:30h	14h	15h	29h
Next Check-in	Not in 23:00-06:59	Not in 23:00-07:59	Not in 23:00-08:59	Not in 23:00-09:59	No Restriction

Table 4.3: Rest and restrictions on check-in time based on previous check-out time

Note that there is no restriction on the check-in times for pairings ending from 05:01, since the 29 hours rest requirement already prohibits the next pairing to start early in the morning.

4.2.2. Crew Rostering

Labor agreements concerning crew rostering are divided in requirements regarding recuperation periods and regulations on free time periods. First we discuss the *recuperation periods*. Recuperation periods are continuous periods of 24 hours in which a crew member cannot work due to recuperation of specific pairings. The recuperation period is positioned directly after the pairing which generates the recuperation period and may always overlap with the rest corresponding with that pairing, but may never overlap with the free time. There are exactly three situations when a recuperation period is generated:

- Three sequential pairings generate a recuperation period if exactly one of these pairings has a check-out time after 03:00. The recuperation period is placed exactly after the pairing which ends after 03:00.
- Two sequential pairings generate a recuperation period if exactly one of these pairings has a check-in time before 02:30. The recuperation period is placed exactly after the pairing which starts before 02:30.
- Three sequential pairings generate a recuperation period if exactly one of these pairings has a check-in time before 03:30. The recuperation period is placed exactly after the pairing which starts before 03:30.

Figure 4.7: Example of a recuperation period within a roster-line

See Figure 4.7 for an example of a recuperation period that is generated by the check-out time of last pairing. A pairing that has the potential to generate a recuperation period is called a *recuperation generating pairing*. Note that not every recuperation generating pairing actually generates a recuperation period.

The next regulations we consider are the regulations on free time periods. As stated earlier, a roster period spans 35 days from which at most 20 are working days and at least 15 are free time days. The weekly working pattern corresponding to this contains (maximum) 4 days of work and 3 days of free time. This free time period contains exactly 72 hours, but the exact starting time of the free time period depends on the last pairing before the free time following these three rules:

- If the last pairing before the free time generated a recuperation period, then the free time period starts exactly after the recuperation period rounded above by the hour (see Figure 4.7).
- If the last pairing before the free time period is a night pairing, but does not generate a recuperation period, then the free time period starts after the rest of the last pairing rounded above by the hour.
- If the last pairing before the free time period has a check-out time between and including 20:01-01:00, then the free time period starts exactly at 05:00.
- If the last pairing before the free time period has a check-out time before and including 20:00, then the free time period starts exactly at 01:00.

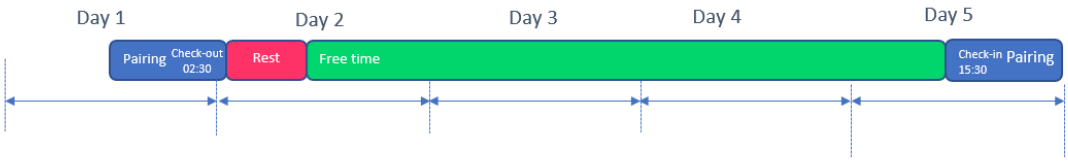

Figure 4.8: Example of a night pairing before free time

In Figure 4.8 is an example of a pairing that ends past 01:00 and therefore affects the position of the free time period. The night pairing before the free time causes the earliest possible check-in time of the pairing past the free time to be late on day 5, which may or may not affect the productivity of the crew as we will show in the Chapter 8.

5. Solution Methods for Pairing and Retiming

As mentioned before, the crew linking document determines vastly how the pairings should be constructed. However, the pairings created in the crew linking document may very well be suboptimal. This information gives rise to create the pairings in a different way than previously determined. This chapter contains solution methods to solve the pairing problem according to the crew linking document as well as solution methods to solve part of the pairing problem differently. Additionally, we solve the pairing problem when we allow for minor changes in the flight schedule. First we describe how we create the pairings via the crew linking document.

5.1. Classical Pairing Creation

The goal of this stage is to assign all the scheduled legs to pairings, such that all the legs are exactly in one pairing and the crew linking document is warranted. First the duty periods are created. Therefore, we take the set of all legs and create all the duty periods that have been preassigned by the crew linking document, which are duty periods containing 4 legs as well as lay-overs. All the legs that are contained in the document are deleted from the leg set and the remaining legs are sorted by aircraft registration. Then the legs are checked for triangle duty periods; these are duty periods that have two stops at a foreign airport before they return to a base. After creating duty periods containing these legs and removing them from the leg set, duty periods are created for legs that return to a different base than where they started and hence they are deleted from the leg set. Finally, all duty periods involving regular outbound and inbound are created and deleted from the leg set. When the leg set is empty, the process is terminated and all the legs have been included in exactly one duty period. The pseudo code of this entire process is given by Algorithm 1.

When all the legs have been assigned to duty periods, duty periods are extended to pairings. Therefore the duty periods have to be extended such that the beginning base is exactly the same as the end base. This process takes two steps.

First the lay-over pairings are completed by extending the duty period that ends at a foreign airport with the duty period that starts at that foreign airport the next day such that the duty period starts and ends at a base. In this way two duty periods can be converted to one two-day pairing. An example of such a lay-over pairing is given by Figure 5.1.

```
Data: Leg set
Result: Duty periods
initialization;
for legs in leg set do
    Check if leg is in leg set;
    if leg in crew linking document then
        Create duty period involving all legs in crew linking document;
        Delete these legs from leg set;
    end
end
Divide remaining leg set into leg set per aircraft registration;
for aircraft registration do
    if triangle duty period recognized in leg set then
        Create duty period involving all legs in triangle sequence;
        Delete these legs from leg set;
    end
    if duty period involving different home bases recognized in leg set then
        Create duty period involving all legs in this duty period;
        Delete these legs from leg set;
    end
    if regular outbound-inbound duty period is recognized in leg set then
        Create duty period involving all legs in this duty period;
        Delete these legs from leg set;
    end
end
```

Algorithm 1: Pseudo code to create duty periods

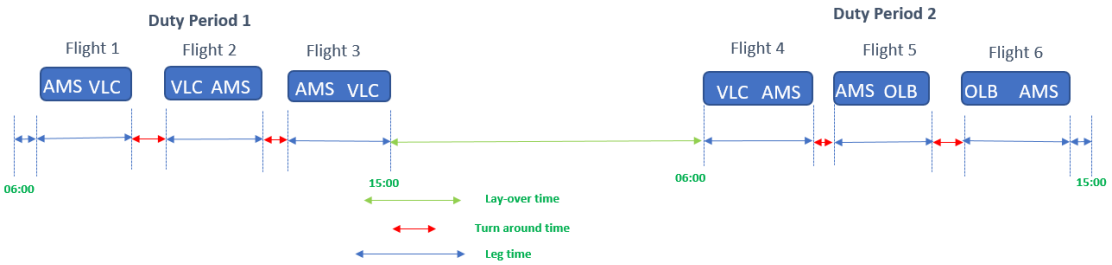

Figure 5.1: Example of a pairing with lay-over

Secondly, every pairing starting at a base has to end at the same base. Therefore a taxi ride has to be appended to every pairing that ends at a different base than where it starts. However, a duty period starting at Amsterdam and ending in Rotterdam can have a crew from Rotterdam being taxied to Amsterdam before the duty period or it can have a crew from Amsterdam being taxied to Rotterdam after the duty period. Generally, we use the following guideline as to where to append a taxi; if in anyway the pairing can become a night pairing because of the taxi, we append the taxi in such a way that the pairing does not become a night pairing. For every duty period that either start or ends in Amsterdam, we append a taxi such that the crew starts and ends in Amsterdam. For a pairing that starts in Eindhoven and ends in Rotterdam or the other way around, we append a taxi such that the pairing starts and ends at Rotterdam. For exact taxi times see Appendix A.2. For an example of a leg set being converted to a pairing see Appendix A.3.

Finally, the pairings are completed by extending the pairing with pairing check-in time which is one hour before the first departure and with pairing check-out time which is half an hour after the last arrival. Additionally, the pairing next check-in time is attached to every pairing based on the rest requirements in Table 4.3, the check-in limitations of Table 4.2 and recuperation periods. Instead of appending recuperation periods after three or two pairings in a row, a recuperation period is scheduled after every recuperation generating pairing, because the difference in results of the schedules we will test are negligible (see Chapter 9).

5.2. Optimal Pairing

In this section we try to optimize part of the pairing problem by ignoring the crew linking document. The pairing part we want to optimize are the link duty periods. These are the duty periods that can be combined with other link duty periods on the same day to form double pairings, which are pairings containing 4 legs. In this way a crew member can do more than two flight legs in one day and therefore enhance his/her productivity. Basically, we want to maximize the number of double pairings under the constraint that the turn around times between the legs are sufficient and the maximum work time does not exceed the labor constraints of Table 4.1, but we allow another prioritization of the variables as well.

More specifically, let $f \in F$ be all the link duty periods in the schedule. The link duty periods contain two legs, an inbound and an outbound leg. Let $P \in \mathcal{P}$ represent a feasible pairing and $\mathcal{P}$ the set that contains all feasible pairings. These pairings can contain either 4 legs (double pairing) or 2 legs (single pairing). We define d_P, dt_P and t_P as the work time, departure time and turn around time respectively for every pairing P. We define the cost of a pairing c_P as 1 for single pairings P and $c_P = \frac{d_P}{d_{max}} + \frac{t_{min}}{t_P} \in [1.5, 2]$ for double pairings. In this way we make sure it is always cheaper to create double pairings instead of two single pairings. The problem can then be formulated as a weighted set covering problem:

$$\begin{aligned} & \min \sum_{P \in \mathcal{P}} c_P x_P & & \\ \text{subject to} \quad & \sum_{P \in \mathcal{P}:\, f \in P} x_P = 1 & & \forall f \in F \\ & x_P \in \{0,1\} & & \forall P \in \mathcal{P}. \end{aligned} \tag{5.1}$$

Contrary to the set covering problem for crew rostering, solving this set covering problem is relatively easy, because we solve the pairing problem per day. This limits the number of feasible pairings considered. Consequently, the number of feasible double pairings is limited and enumerable. Note that it is possible that the optimal solution does not provide the maximum number of double pairings, since we work with a cost function per pairing. In order to get the maximum number of double pairings, we take cost function $c_P = 1$ for every double pairing P.

5.3. Retiming

In addition to the traditional pairing problem, we try to solve another pairing problem where we allow for minor changes in the schedule. Making minor changes in the departure and arrival time of a flight can ensure that certain links become feasible, because the minimum turn around time between flights is slightly modified or because the total pairing time does not exceed the maximum turn around time. An example of an unfeasible pairing becoming feasible by retiming is given by Figure 5.2. By retiming the AMS-OLB pairing 10 minutes later, the turn around time in AMS indicated by the black arrow extends and reaches its minimum required. In this way the pairing becomes feasible. Although this problem can also be formulated as a set covering problem, the number of feasible pairings increases rapidly when we allow for changes in departure time. Therefore, we introduce a new method to solve this pairing problem.

Let $f \in F$ be all the link duty periods in the schedule and create for each link duty period 3 copies f^1, f^2 and f^3 where each copy represents the same link duty period with a minor change in arrival and departure time. We create an undirected graph $G = (V, E)$ where the nodes are all the copies of the link duty period, f^1, f^2 and f^3 for all $f \in F$. In addition to these, we create control nodes f^4 and f^5 for all $f \in F$. Edges (f^i, f^j) are created within the same link duty period nodes for all $i \in \{1, 2, 3\}$, $j \in \{4, 5\}$ and for all $f \in F$ with artificial high weight $w_e = c$. Within different link duty period nodes, we

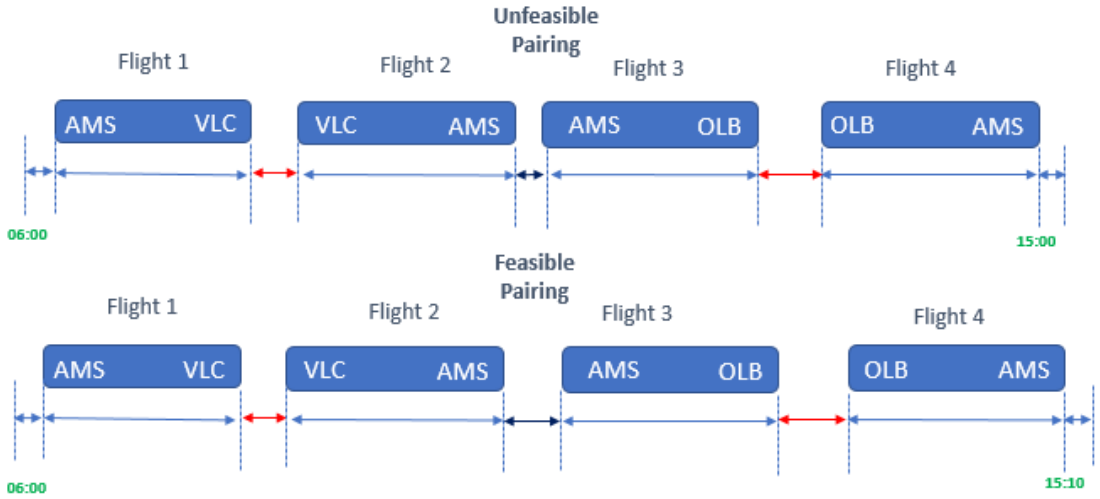

Figure 5.2: Example of unfeasible pairing, becoming feasible by retiming

only create edges between the nodes if the corresponding double pairing of the node is feasible. Note that the feasibility depends on the copy version of the link duty period nodes, since the copies differ in departure and arrival time. Formally, for all $f, g \in F$ and $1 \leq i, j \leq 3$, edge (f^i, g^j) is created if and only if f^i and g^j is a feasible double pairing. The weight on edge e is $w_e = \frac{1}{c_P}$ where c_P is the cost of double pairing P defined in Section 5.2. For an example of such a graph see Figure 5.3.

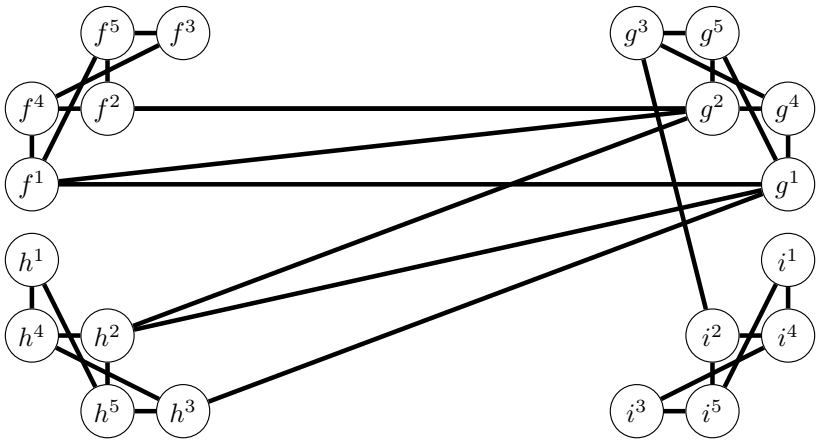

Figure 5.3: Example of an unmatched graph with 4 link duty periods $f, g, h, i \in F$

Now a maximum weight maximum cardinality matching in G is created. The matching gives the copy versions of the link duty period that are used as well as the double pairings that are created given the considered weight. The maximum cardinality property guarantees that the maximum number of double pairings is formed. The artificial

cost c imposes that we can only match one of the copies of a pairing but not multiple copies. The matching is a maximum cardinality matching, but often not a perfect or near-perfect matching. This is because often a number of nodes can only be matched with one other node, but only one of these nodes can be matched leaving the other nodes exposed. This is also visible in the maximum weight maximum cardinality matching M in Figure 5.4. Note that node f^2 and i^2 are M-exposed. The artificial high weights on the edges within the nodes of link duty period g are such that at most one link duty copy nodes of g can be matched, which leaves the nodes f^2 and i^2 M-exposed.

Similarly as the set covering problem in Section 5.2, besides finding the maximum number of double pairings, it can also be interesting to find those double pairings that sum up to the highest total weight. Therefore, we also make a maximum weight matching, while not imposing maximum cardinality. Note that the given structure of the control nodes still ensures that at most one copy of a link duty period can be matched.

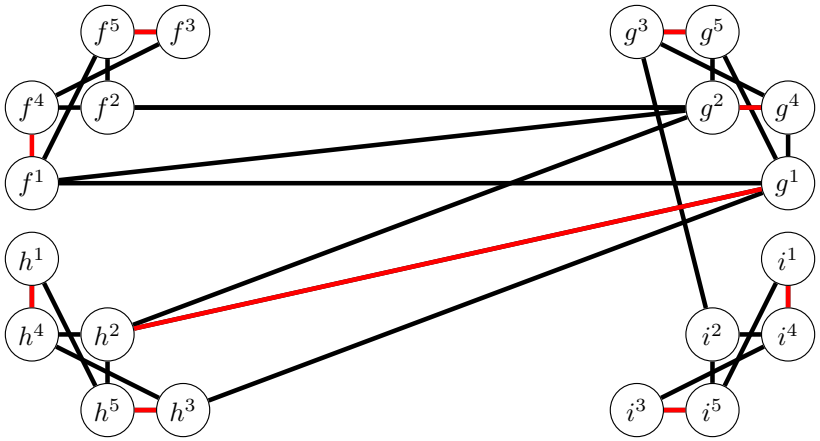

Figure 5.4: Example of a maximum weight maximum cardinality matching.

6. Motivation of Solution Methods for Crew Rostering

In order to better understand the magnitude of the problem, past planned schedules are observed and analyzed for the impact of the night pairings. The schedules concerned are cockpit crew schedules from the entire year from 2014 to 2017. We distinguish different features of night pairings and we count how much each of these features occur and how this affected the productivity consequently. Eventually, these effects are converted to a loss of productivity in hours. We will see that the determination of the loss of productivity gives rise to a certain approach for a solution method discussed in Chapter 7. The three main features of night pairings we distinguish are night pairings that generate recuperation periods, night pairings that generate 29 hours of rest and night pairings that are scheduled before the free time period and therefore affect the productivity.

6.1. Night Pairings with 29 hours rest

As described in Chapter 4, the rest associated with a pairing depends on the check-out time. In particular, pairings that have a check-in time before 23:59 and check-out time after 05:00 have an associated rest period of 29 hours (see Table 4.3). In Figure 6.1 is presented how frequently this occurs per year.

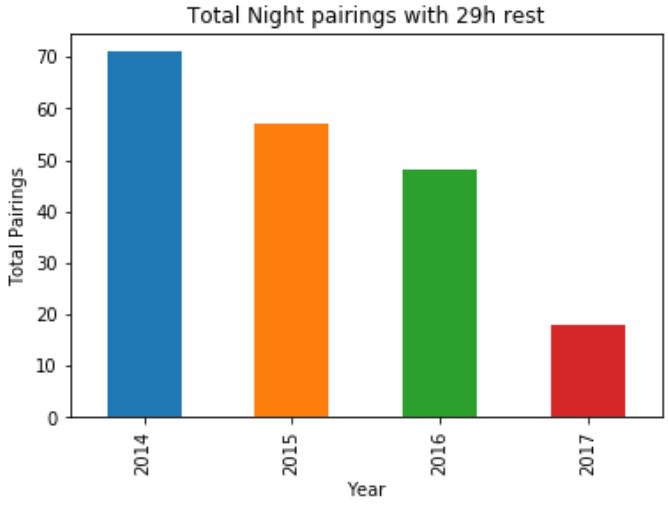

Figure 6.1: Frequency of 29h rest pairings

The loss in associated productivity can be described as the number of hours of extra rest due to these pairings.

6.2. Recuperation

Although there are many pairings that can generate a recuperation period, very few recuperation periods are actually generated. This is because the actual roster system Tbids uses various tricks to prevent recuperation periods from being generated. One of these tricks is to schedule a non-flight duty such as a stand-by between flight duties, such that the recuperation generating pairings are not sequential anymore. In Figure 6.2 is indicated how many recuperation periods are assigned. The rostering system Tbids was introduced in 2016 and the use of these tricks of this system is clearly visible in Figure 6.2.

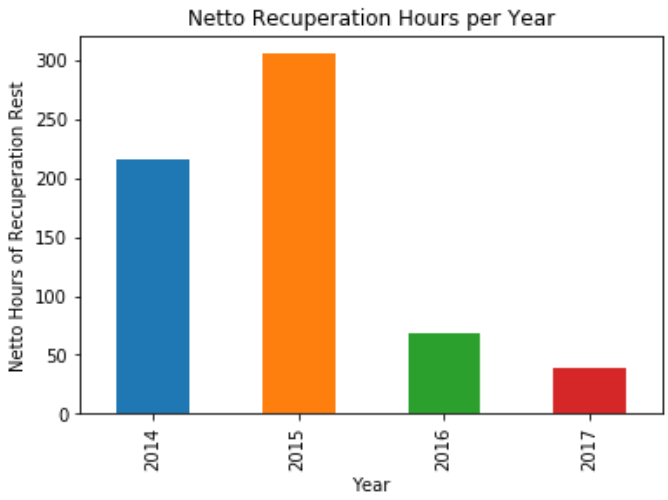

Figure 6.2: Frequency Recuperation Periods

In order to estimate the loss in crew productivity due to recuperation generating pairings, we subtract the mandatory rest requirements from the recuperation period, since the rest requirement is associated with every pairing.

6.3. Night Pairings before Free Time

In the last section we consider the night pairings that are positioned before the free time period, which subsequently affects the position of the free time period (see Figure 4.8). In Figure 6.3 is indicated how frequently this occurred.

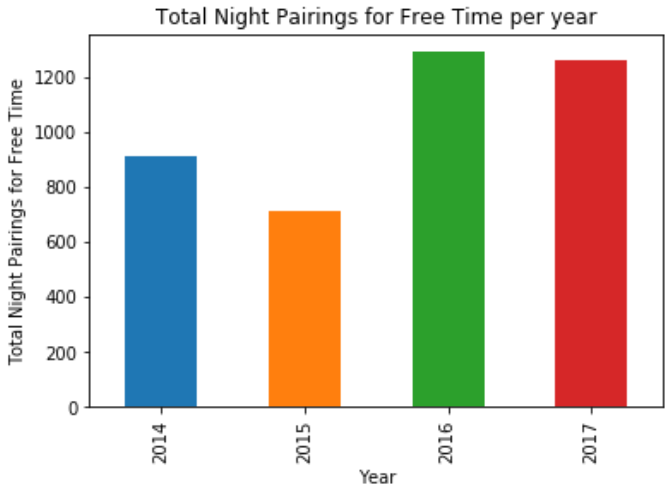

Figure 6.3: Frequency night pairings for free time

Unlike recuperation periods, night pairings have been increasingly assigned before the free time period since the introduction of the new rostering system Tbids in 2016. In contrast to normal pairings before the free time, night pairings cannot have their associated rest overlapped with the free time period. We try to estimate the loss in productivity caused by this feature of night pairings by projecting the average overlap of normal pairings with the free time period to night pairings multiplied by the frequency of night pairings before the free time.

6.4. Conclusion

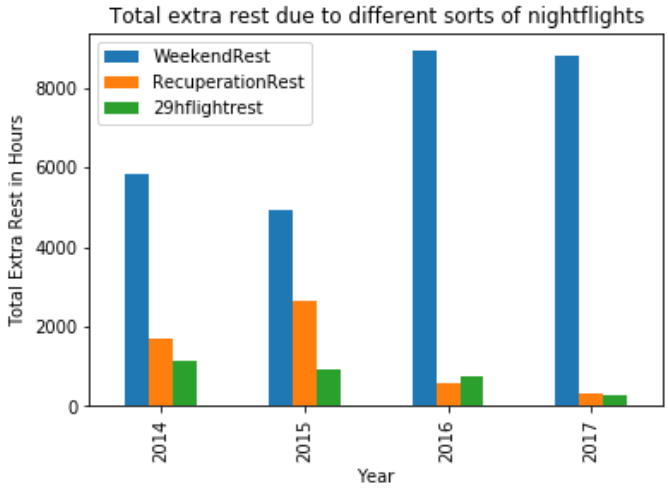

Figure 6.4: Total hours of unavailability due to night pairings

The total estimated loss of productivity in hours is given in Figure 6.4. Important to note is that the loss in productive hours is a loss in available hours of the crew member. That is, the hours indicated by Figure 6.4 are hours that crew members could not work due to the effects of night pairings, but that does not necessarily mean that the crew members would have worked if they were available. This completely depends on whether or not there is a pairing available for this crew member and in fact it is likely that the negative effects of night pairings have been positioned in the part of the roster where a loss in availability will not cause a loss in productivity. This is why this data analysis does not provide enough insight in the impact of night pairings on the productivity and calls for a solution method that estimates the crew productivity within the context of the entire schedule.

7. Solution Methods for Crew Rostering

The goal is to create a model that estimates the crew productivity based on the number of night pairings. Chapter 6 has illustrated that it is hard to calculate the impact per night pairing, because this impact is related to the rest of the pairings. Therefore, the impact of night pairings is calculated within the context of the complete flight schedule. In order to find out what part of the crew productivity is due to night pairings, we test on a variety of schedules, where some schedules involve many night pairings and other schedules involve fewer night pairings. In this way the difference in productivity defines the impact of the night pairings within a certain schedule.

We use certain *key performance indicators* (KPIs) that define the productivity of the crew. The most dominant KPI is the minimum number of crew members necessary such that a certain schedule is feasible, that is, such that all the pairings can be performed at least by one crew member and the labor regulations are warranted. However, it can happen that n pairings can be performed by m crew members, but $n+1$ pairings can also be performed by m members. In this case the crew is more productive, but this is not indicated by the first KPI. Therefore we add two additional KPIs. First we calculate the *full time equivalent* (FTE), which is the total amount of time required to realize the schedule including idle time relative to the total possible work time. Next we calculate the *total idle time* (TID), which is the amount of time a crew member is available but does not work. This can be either because no pairing is available or because the corresponding crew member is currently in its free time or recuperation period. Note that the TID and FTE are interchangeable as one can easily calculate the total amount of idle time by calculating the necessary FTE, multiply it by the total amount of possible work time and subtract the total amount of work and rest time. In Figure 7.1 an example of the concepts of idle time and full time equivalent is given. The length of the long red arrow divided by the total length of the blue arrows is the FTE and the total length of all the green arrows is the TID.

The problem can be solved in several stages. First, we use one of the methods described in Chapter 5 to create feasible pairings. All these pairings together make up the *pairing schedule*, a timetable of pairings and associated rest periods. Secondly, the minimum

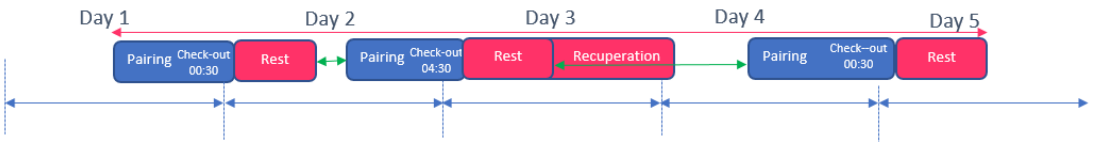

Figure 7.1: Example of idle time and full time equivalent

number of crew members necessary to perform a certain pairing schedule is calculated. Then, given the minimum number of necessary crew members, the problem is optimized for the minimum amount of FTE and total idle time. The resulting KPIs determine the productivity of the crew. In this chapter we present several methods to optimize for the minimum number of crew members required and the minimum amount of idle time and FTE. Additionally, a method is proposed to check which pairings are dropped when we maximize the amount of block time.

7.1. Set Covering Approach

The first problem that needs to be solved is to calculate the minimum number of crew members necessary to realize a certain schedule. A standard way to tackle this problem is the set covering approach. Let F be the set of pairings that need to be covered and let $\mathcal{P}$ be the set of the feasible roster-lines. A roster-line is a sequence of pairings conform law-regulations and labor requirements, hence for all $f \in F$, there exists a $P \in \mathcal{P}$ such that $f \in P$. Let x_P be the decision variable for each roster-line $P \in \mathcal{P}$. That is, the decision variable x_P has value 1 if roster-line P is assigned to a crew member and has value 0 otherwise. The ILP can then be formulated as follows.

$$\begin{aligned} & \min \sum_{P \in \mathcal{P}} x_P & & \\ \text{subject to } & \sum_{P \in \mathcal{P}:\, f \in P} x_P \geq 1 & & \forall f \in F \\ & x_P \in \{0,1\} & & \forall P \in \mathcal{P} \end{aligned} \tag{7.1}$$

The objective function of the ILP aims to minimize the number of crew members necessary to realize the schedule where the first constraint ensures that every pairings is performed at least once. When a pairing is contained in multiple roster-lines, one can always take the subset of the roster-line to obtain an exact solution. The second constraint ensures that we have an integral solution.

The problem with this solution method is that the number of feasible roster-lines increases exponentially when the number of pairings considered increases. Taking into account all the regulations and labor requirements that should be checked for each roster-line, enumerating all roster-lines and solving the ILP becomes prohibitive.

7.2. Basic Minimum Cost Flow Model

In this section we try an alternative approach based on graph theory. At first, we assume that crew members have no days off or free time periods and can work continuously as long as the labor and law requirements (except those concerning days off and free time) are satisfied. Later we introduce a method that includes days off and free time along with the concerning labor and law requirements.

All pairings $f \in F$ have a check-in time f_1 and check-out time f_2. We introduce a digraph $G = (V, A)$. The node set V can be written as a disjoint union $V = V_{F_1} \cup V_{F_2} \cup$

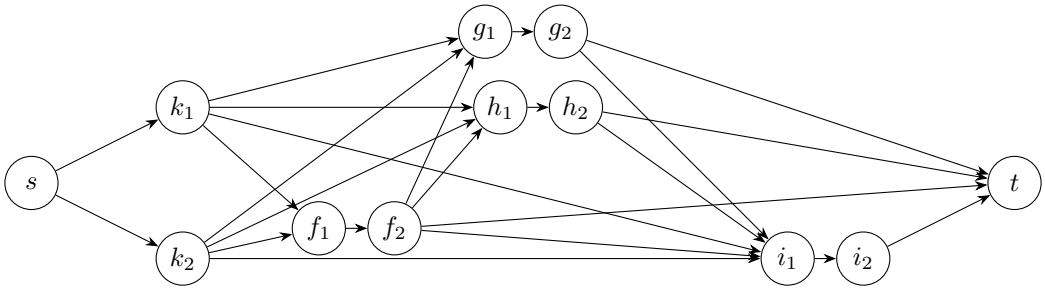

Figure 7.2: Basic Network with $|K| = 2$ and $f, g, h, i \in F$

$V_K \cup \{s\} \cup \{t\}$ where V_{F_1} and V_{F_2} contain the pairing check-in and check-out time nodes f_1 and f_2 respectively for all $f \in F$. V_K consists of the crew nodes $k \in K$ where $|K|$ is the maximum number of start crew members. Additionally, we add a source node s and sink node t to V. Hence $|V| = 2|F| + |K| + 2$.

For every crew member $k \in K$, we create an arc (s, k) from the source to the crew nodes. For every crew member $k \in K$ and every pairing $f \in F$, we add arcs (k, f_1) from all crew nodes to all pairing check-in nodes. Similarly, for every pairing $f \in F$ we create arcs (f_2, t) from pairing check-out nodes to the sink node. We create arcs (f_1, f_2) for all $f \in F$, between all the check-in nodes to the check-out node of the corresponding pairing. Finally, for every pair of pairings, $f, g \in F$, we create arcs (f_2, g_1) if the law regulations and labor requirements allow pairing g to be sequential to pairing f. For an exact overview about the law and labor rules concerning pairings allowed to be sequential to each other, see Section 4.2. The total number of arcs then becomes $|A| = |V_K| + |V_K| \cdot |F| + 2|F| + |S|$, where the set S contains the arcs that connect pairings to other pairings. This number is unknown until all pairing combinations are checked for feasibility, but an upper bound is given by $|F| \cdot |F - 1|$, since no pairing is connected to itself. An example of such a network is given by Figure 7.2, where the network is restricted to 2 crew members and 4 pairings.

7.2.1. Minimum Crew Required

The first objective we have is to find out the minimum number of crew members necessary to perform the schedule. We append the nodes with supply data s_v where s_v has value $|K|$ if $v = s$ and s_v has value $-|K|$ if $v = t$. All the other nodes have supply 0. The arcs are appended with costs, minimum capacity and maximum capacity values (c, u_{min}, u_{max}). Because we want to minimize the number of crew necessary, we add arc values $(c, u_{min}, u_{max}) = (1, 0, 1)$ to $a = (s, k)$ for all $k \in K$. In order to ensure that every pairing is covered, we add values $(c, u_{min}, u_{max}) = (0, 1, 1)$ to $a = (f_1, f_2)$ for all $f \in F$. The rest of the arcs are appended with values $(c, u_{min}, u_{max}) = (0, 0, 1)$. We introduce flow variable x_a for which $u_{min}(a) \leq x_a \leq u_{max}(a)$. Now in order to minimize the number of crew members necessary to perform a schedule, the problem

can be formulated in a way that resembles the minimum cost flow problem:

$$\begin{aligned} & & \min \sum_{a \in A} & \; c_a x_a & & \\ \text{subject to} & & x_a & \leq \; u_{max}(a) & & \forall a \in A \\ & & x_a & \geq \; u_{min}(a) & & \forall a \in A \\ & \sum_{a \in \delta^+(v)} x_a - \sum_{a \in \delta^-(v)} & x_a & = \; s_v & & \forall v \in V \backslash \{s,t\} \end{aligned} \tag{7.2}$$

Because the only arcs appended with non-zero costs are the arcs from the source node to the crew nodes, the objective function minimizes the number of crew members required. The source and the sink node are excluded for having flow conservation, because the maximum number of crew members equals the supply value, but we do not want to force the program to use the maximum amount, since we are actually optimizing for the minimum number of crew members we need. In Figure 7.3 the flow corresponding to an optimal solution of linear program (7.2) is given.

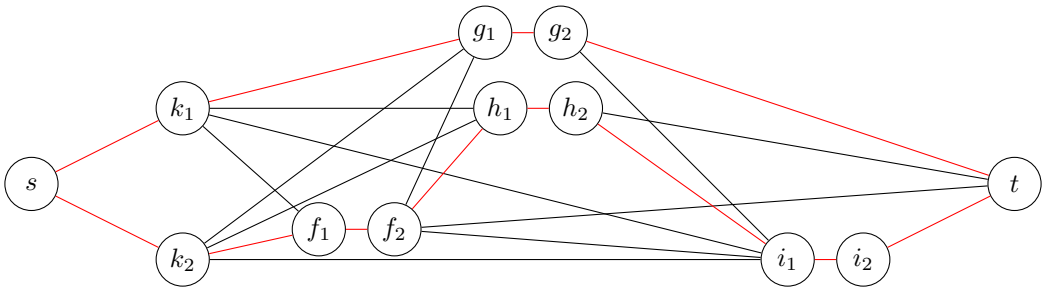

Figure 7.3: Flow of an optimal solution of the network in Figure 7.2 indicated in red

This formulation is not an exact minimum cost flow network since the flow conservation constraints do not apply for nodes s and t. However, one can easily work around this by allowing an arc from t to s and impose flow conservation on s and t. This guarantees that there is an integral optimal solution. Also, one can force fixed supply and demand values on s and t and reduce these values when the problem contains a feasible solution. Finding a feasible solution in a minimum cost flow problem can be solved by finding a maximum flow in the network. This can be solved in polynomial time by Edmonds–Karp algorithm [5].

7.2.2. Minimum TDI and FTE

The linear program stated in (7.2) gives a solution to the first KPI, namely the minimum numbers of crew members required to realize a schedule. In order to obtain the second and third KPI, a few changes should be made to the digraph:

- The number of crew nodes is reduced to the output value of the former linear program (7.2), which is the minimum number of crew members necessary.

- For all $k \in K$, arcs (s, k) from the source node to the crew nodes have cost 0 instead of cost 1.
- For all arcs from the check-out pairing nodes to check-in pairing nodes, we add cost $Z(f, g)$ where $Z(f, g)$ is the idle time from the check-out time of pairing f plus the associated rest of pairing f to the check-in time of pairing g. Note that $Z(f, g)$ can only be positive since the arcs are created in such a way that the labor agreements allow pairing g to be sequential to pairing f.

Now with a slightly modified formulation of linear program (7.2), we can create a roster where we minimize the total amount of idle time between pairings given a fixed number of crew members:

$$\begin{aligned} & & \min \sum_{a \in A} & c_a x_a & & \\ \text{subject to} & & x_a & \leq & u_{max}(a) & \quad \forall a \in A \\ & & x_a & \geq & u_{min}(a) & \quad \forall a \in A \\ & \sum_{a \in \delta^+(v)} x_a - \sum_{a \in \delta^-(v)} & x_a & = & s_v & \quad \forall v \in V. \end{aligned} \tag{7.3}$$

Including the source and the sink node in the flow conservation constraint ensures that the exact number of crew members is used as the previous model showed to be required. Also, because we impose the conservation constraint on every node, we now have a proper minimum cost flow problem. Integrality constraints are redundant by Theorem 3.1 and therefore omitted.

Note that the amount of idle time between pairings is minimized and not the idle time from the beginning time of the schedule to the check-in of the first pairing or the idle time from the last pairing to the end of the schedule time. The reason for this is of course that the total amount of idle time in the latter case is fixed, since it will be a function of the number of crew members and the sum of the pairing times with their associated rests.

When the total idle time is low, it means that the pairings follow each other up properly and even though this does not decrease the number of crew members needed in the optimal crew rostering schedule, it can make all the difference in the gross crew rostering schedule. If we take the minimum amount of total idle time and add the total amount of working time with their associated rest time, we obtain the minimum amount of FTE required to perform the schedule. Similar to the minimum total idle time, the FTE can make a major difference in the gross schedule.

7.3. Extended Minimum Cost Flow Model

In order to tackle the problem including the free time periods, we extend the model by creating layered graphs. First the basic model is presented. Then is described how the model is adapted specifically to tackle the different problems separately.

7.3.1. Basic Model

A roster period for a crew member contains 35 days. These 35 days contain at most 20 working days and 15 days off. This corresponds to the traditional working pattern where crew members work at most four days and have three days off. Sometimes the working pattern is slightly changed to 5 working days and 2 days off and then to 3 working days and 4 days off as long as the total number of working days per roster period does not exceed 20 days. In this chapter we only consider traditional workweeks of (at most) 4 days working and 3 days off.

We define layered directed graphs G_1, G_2, G_3 and G_4 where $G_i = (V_i, A_i)$ for $1 \leq i \leq 4$ represents a layer. For every layer, we define the node set in that layer as $V_i := V_1^i \cup V_2^i$ where the nodes in V_1^i and V_2^i still correspond to the check-in and check-out time of all pairings $f \in F$. Note that we have 4 copies of the exact same pairing in the 4 different layers. Additionally, for every $f \in F$ and $i \in \{1, 2, 3, 4\}$, we add arcs (f_1^i, f_2^i) to A_i with arc data $(c, u_{min}, u_{max}) = (0, 0, 1)$.

Arcs are constructed between the layers and between the different pairing nodes as before in the following way.

- For all $f, g \in F$ for which labor requirements allow pairing g being sequential to pairing f and the check-in of pairing g is exactly one day after the check-in of pairing f, create arc (f_2^i, g_1^{i+1}) for $i \in \{1, 2, 3\}$. For an example of such an arc see Figure 7.4.

- For all $f, g \in F$ for which labor requirements allow pairing g being sequential to pairing f and the check-in of pairing g is at most two days after the check-in of pairing f, create arc (f_2^i, g_1^{i+2}) for $i \in \{1, 2\}$.

- For all $f, g \in F$ for which labor requirements allow pairing g being sequential to pairing f and the check-in of pairing g is at most three days after the check-in of pairing f, create arc (f_2^1, g_1^4).

This construction of different layers keeps track of the working days of a crew member. If pairing f is followed by pairing g and the starting day of g is exactly one day after the starting time of f, then the arc connecting these pairings goes one layer up as the working days before the free time period decreases by one.

Now in order to go back to working day one, there is a free time period that should be bridged first, but the start of the free time period depends on the last pairing performed before the free time period. Generally, a free time period lasts 72 hours, but there are several labor regulations that define more specifically when the free period starts and what it can overlap. For an exact overview of these rules, see Section 4.2.

We create a function $Y(f, g)$ for $f, g \in F$ that determines whether or not we can connect two different pairing nodes while bridging the free time period: from all arcs from pairing check-out nodes in G_4 to check-in pairing nodes in G_1 we create arcs (f_2^4, g_1^1) if and only if the idle time conform the free time period regulations, $Y(f, g)$, exceeds 72 hours. For an example of such an arc see Figure 7.5.

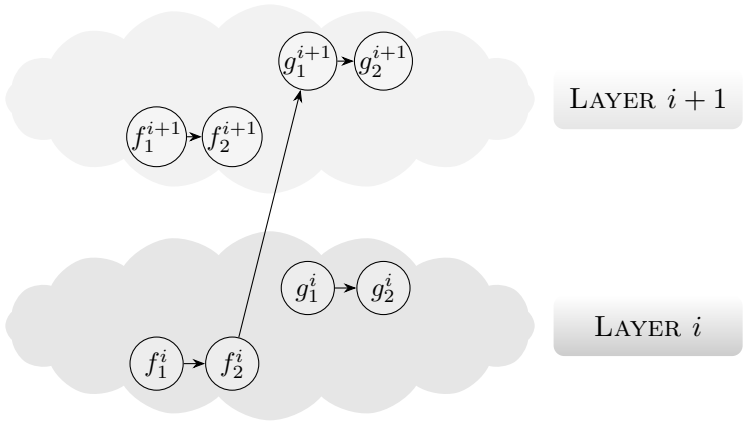

Figure 7.4: Example of an arc between two layers i and $i+1$ for $i \in \{1, 2, 3\}$ and $f, g \in F$.

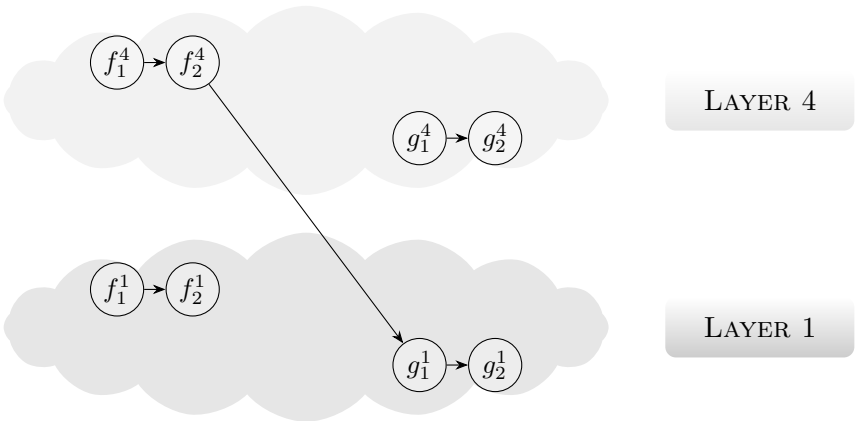

Figure 7.5: Example of an arc between layer 4 and layer 1 bridging a free time period.

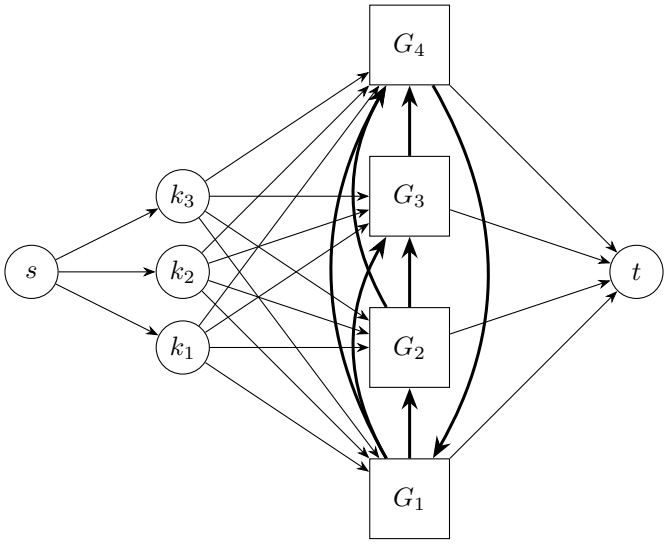

Figure 7.6: Example of the complete network with pairing nodes contained in the layers

Then we add $|K|$ crew nodes and for all $k \in K$, $f \in F$ and $i \in \{1, 2, 3, 4\}$, we add arcs (k, f_1^i) from each crew node to every pairing check-in node in every layer[1]. Additionally, we add source and sink nodes s and t and for all $k \in K$ we create arcs (s, k) from the source node to all crew nodes. Similarly, for every $f \in F$ and every $i \in \{1, 2, 3, 4\}$ we create arcs (f_2^i, t) from all check-out pairing nodes in every layer to the sink node[2]. Again, the nodes are appended with supply value s_v where s_v has value $|K|$ if $v = s$ and s_v has value $-|K|$ if $v = t$. All the other nodes have supply value 0. In Figure 7.6 is the complete graph structure outlined, but the pairing nodes are contained in the layers. Each thick arc represents multiple arcs between nodes in the different layers.

7.3.2. Minimum Crew Required

First we use this basic model to calculate the minimum number of crew members necessary to perform a schedule taking into account the free time periods that crew members have. Arc values are defined in the following way:

- All arcs created in the basic model are appended with arc value $(c, u_{min}, u_{max}) = (0, 0, 1)$, **except for**
- All arcs (s, k) from the source node to all the crew nodes $k \in K$, which we append with arc value $(c, u_{min}, u_{max}) = (1, 0, 1)$.

Note that similar to the graph used to determine the minimum number of crew members necessary without the free time requirement, the only arcs with non-zero costs are the arcs running from the source node to the crew nodes. However, the major difference

[1] Assuming every crew member can start at any working day.
[2] Assuming every crew member can end at every working day.

between this model and the model before is that the minimum flow in the arcs from the check-in pairing nodes to the check-out pairing nodes is 0 and not 1 as before. This is because we do not have unique pairing nodes, but we have four identical pairing nodes positioned in the different layers in the graph and exactly one of the four pairing nodes should be covered. To ensure that this constraint is satisfied, we formulate the linear program in the following way:

$$
\begin{aligned}
& & \min \sum_{a \in A} & \; c_a x_a & & \\
\text{subject to} & & x_a & \leq \; u_{max}(a) & & \forall a \in A \\
& & x_a & \geq \; u_{min}(a) & & \forall a \in A \\
& \sum_{a \in \delta^+(v)} x_a - \sum_{a \in \delta^-(v)} & x_a & = \; s_v & & \forall v \in V \backslash \{s,t\} \\
& \sum_{i=1}^{4} \sum_{f \in F_i} & x_{f_1^i f_2^i} & \geq \; 1 & & \forall f \in F \\
& & x_a & \in \; \{0,1\} & & \forall a \in A.
\end{aligned}
\tag{7.4}
$$

The second constraint can be simplified to $x(a) \geq 0$ since $u_{min}(a) = 0$ for all $a \in A$. The fourth constraint ensures that at least one of the copies of the pairing is covered, but it also increases the complexity of the problem. Now the problem is not a minimum cost flow problem anymore, hence integrality constraints should be included. This means that the problem should be tackled with an integer linear program solver. Note that that it does not matter if a pairing is covered multiple times since we can always take the sub roster-line and the solution remains feasible.

7.3.3. Minimum TDI and FTE

After the previous problem is solved, one obtains the minimum number of crew members necessary to perform a certain pairing schedule. Now we want to minimize the total amount of idle time time given a fixed number of crew members. We make the following changes to our digraph:

- For all arcs of the form (f_2^i, g_1^{i+1}), (f_2^i, g_1^{i+2}) and (f_2^i, g_1^{i+3}) for $f, g \in F$ and $i \in \{1,2,3\}$, $i \in \{1,2\}$ and $i = 1$ respectively, replace arc values by $(c, u_{min}, u_{max}) = (Z(f)(g), 0, 1)$ where $Z(f)(g)$ is the idle time value from pairing f to pairing g defined in Subsection 7.2.2.
- For all $f, g \in F$ for which arcs exist between (f_2^4, g_1^1), replace arc values by $(c, u_{min}, u_{max}) = (Z(f)(g), 0, 1)$.
- For all arcs (s, k) from the source node to the crew nodes for $k \in K$, replace arc values by $(c, u_{min}, u_{max}) = (0, 0, 1)$.

Even though we use the function $Y(f, g)$ to calculate whether or not an arc should be created between pairings bridging free time periods, the function calculates the idle time

depending on the last pairing before the free time. This means that for a night pairing, the idle time starts after the rest period, while the idle time for day pairings can overlap with the rest period. However, we need a universal definition of idle time regardless of the additional features of the last pairing before free time. Therefore, we make use of the earlier defined function Z where $Z(f)(g)$ describes the total time between the check-out of pairing f with associated rest and the check-in time of pairing g. The linear program formulation then becomes:

$$\begin{aligned}
& & \min \sum_{a \in A} & c_a x_a & & \\
\text{subject to} & & x_a & \leq & u_{max}(a) & \forall a \in A \\
& & x_a & \geq & u_{min}(a) & \forall a \in A \\
& \sum_{a \in \delta^+(v)} x_a - \sum_{a \in \delta^-(v)} & x_a & = & s_v & \forall v \in V \\
& \sum_{i=1}^{4} \sum_{f \in F_i} & x_{f_1^i f_2^i} & \geq & 1 & \forall f \in F \\
& & x_a & \in & \{0,1\} & \forall a \in A.
\end{aligned} \tag{7.5}$$

7.4. Column Generation

We follow the column generation approach for crew scheduling of Borndörfer and Schelten [2]. Assuming we have a directed graph $G = (V, A)$ where the node set V consists of the pairings $f \in F$ that need to be covered and the arc set A contains the links that connect the pairings. Also, there is a source node s and a sink node t such that a roster line can be viewed as a path P from s to t. The links are labeled with costs c_a that indicate the idle time of a crew member from one pairing to another. We apply a positive penalty cost c_a when using a crew member for arcs of the form (s, f) for all $f \in F$. A path P from s to t has costs $c_P = \sum_{a \in P} c_a$. We introduce decision variables x_P for each path $P \in \mathcal{P}$. The goal is to minimize the number of necessary crew members and total amount of idle time given that every pairing is covered at least once. The problem can be stated as:

$$\begin{aligned}
& \min \sum_{P \in \mathcal{P}} & c_P x_P & & \\
\text{subject to} & \sum_{P \in \mathcal{P}:\, f \in P} & x_P & \geq & 1 & \forall f \in F \\
& & x_P & \in & \{0,1\} & \forall P \in \mathcal{P}.
\end{aligned} \tag{7.6}$$

In order to apply column generation we first reformulate the set covering problem where we introduce the pairing path incidence matrix $B := (b_{fP})$ where $b_{fP} = 1$ if pairing $f \in P$ and $b_{fP} = 0$ otherwise. Hence a column P of matrix B corresponds to the incidence vector of the set of pairings in P. Let $c \in \mathbb{R}_+^{\mathcal{P}}$ and $x \in \mathbb{R}^{\mathcal{P}}$, then the problem can be reformulated as a set covering problem:

$$\min c^T x \quad \text{subject to} \quad Bx \geq \mathbb{1} \quad x \in \{0,1\}^{\mathcal{P}}.$$

7.4.1. Column Generation Algorithm

The linear relaxation of the set covering problem can be formulated as:

$$\min c^T x \quad \text{subject to} \quad Bx \geq \mathbb{1} \quad 0 \leq x \leq \mathbb{1}.$$

The dual of the relaxation of the set partitioning problem is:

$$\max \sum_{i \in V} \pi_i \quad \text{subject to} \quad B^T \pi \leq c \quad \pi \geq 0.$$

We start with an initial solution $B' = B_{\mathcal{P}'}$ for a subset of roster lines $\mathcal{P}' \subset \mathcal{P}$, which is feasible. Associated with B' are the costs c' vector and the decision variable $x' \in \mathbb{R}^{\mathcal{P}'}$. By solving the restricted master LP (RMLP)

$$\min c'^T x' \quad B'x' \geq \mathbb{1} \quad 0 \leq x' \leq \mathbb{1} \tag{RMLP}$$

one obtains dual solution π. With the dual solution one can compute the reduced cost of a path by $\bar{c_P} := c_P - \pi^T B$. The goal will be to find the path with the most negative reduced cost and add this one to the initial solution, hence the pricing problem becomes:

$$\min_{P \in \mathcal{P}} \bar{c_P}. \tag{Pricing}$$

The reduced cost of a path can be decomposed into the sum of the reduced cost of all the arcs in the path where the reduced cost of an arc (i, j) can be formulated as

$$\bar{c_{ij}} := c_{ij} - \pi_i.$$

In this way the pricing problem becomes a shortest path problem with possibly negative costs but without negative cycles:

$$\begin{aligned} & \min \sum_{a \in A} \quad \bar{c_a} x_a & & \\ \text{subject to} \quad & \sum_{a \in \delta^{out}(v)} - \sum_{a \in \delta^{in}(v)} \quad = \quad \begin{cases} 1 & \text{if } v = s \\ -1 & \text{if } v = t \\ 0 & \forall v \in V \backslash \{s, t\} \end{cases} & & \text{(Pricing)} \\ & x_a \quad \in \quad \{0, 1\} \qquad \forall a \in A. & & \end{aligned}$$

Because the shortest path problem allows the costs to be negative, we solve the pricing problem with the Bellman Ford algorithm. The idea will be the following: first the restricted master problem will be solved and dual variables π will be generated. Then the dual variables are used to solve the pricing problem. The solution of the pricing problem is a new column that can be added to the restricted master problem if its reduced cost is negative. In this way the process repeats until no path in the pricing problem can be found with negative reduced cost. If that is the case, the optimal solution of the restricted master problem does not have to be integer. In order to continue a branching strategy should be implemented.

7.4.2. Branching strategy

In the classical branching strategy, one looks at the fractional component x_P for which $0 < x_P < 1$ and branches into two new nodes, one where we impose the restriction $x_P = 0$ and one where $x_P = 1$. However, in the context of the set cover problem, this technique generates a very unbalanced growth of the branch and bound tree. That is, the restriction $x_P = 0$, which simply means not using a certain path, has very little effect since there are many available paths very similar to the path that is branched upon. On the other hand $x_P = 1$ imposes a radical restriction on the solution by forcing a certain path in the solution while many similar paths might not have been checked. Therefore, Ryan and Foster [15] call for a more balanced branching technique based on the following theorem:

Theorem 7.1. *[15] Let B be an $n \times m$, $0-1$ matrix and let the solution $Bx = \mathbb{1}$ be fractional. That means that there exists at least one j for which $0 < x_j < 1$. Additionally, assume that all columns of B are distinct and non-zero. Then there exist two rows, i and i' such that*

$$0 < \sum_{k:b_{ik}=b_{i'k}=1} x_k < 1.$$

Proof. Consider some j for which $0 < x_j < 1$ and consider any row i for which $b_{ij} = 1$. Since $\sum_{k=1}^{m} b_{ik}x_k = 1$ and we considered fractional variable x_j, we know that there is another column j' such that $0 < x_{j'} < 1$ and $b_{ij'} = 1$. Consider now a row i' for which $b_{i'j} = 1$ or $b_{i'j'} = 1$, but not both. This row exists because otherwise we would have a duplicate column in the matrix. Using this information we know:

$$\begin{aligned} 1 &= \sum_{k=1}^{m} b_{ik}x_k \\ &= \sum_{k:b_{ik}=1} x_k \\ &> \sum_{k:b_{ik}=b_{i'k}=1} x_k \\ &> 0 \end{aligned}$$

where the strict inequality follows from the fact that only j or j' is included in the last summation where the previous summation contains both. □

The branching strategy resulting from this theorem is that we branch on

$$\sum_{k:b_{ik}=b_{i'k}=1} x_k = 0 \quad \text{or} \quad \sum_{k:b_{ik}=b_{i'k}=1} x_k = 1.$$

In the context of our crew scheduling problem, this strategy basically means that we branch on whether pairings will be covered by the same path or different paths. When

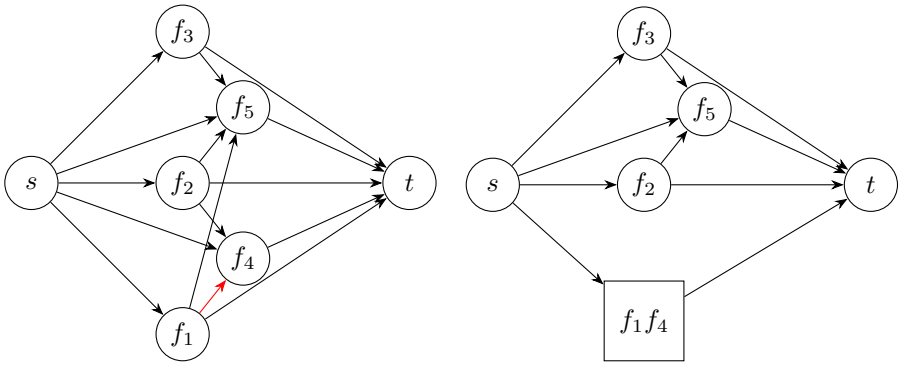

Figure 7.7: Example of how the digraph changes when $\sum_{k:b_{1k}=b_{4k}=1} x_k$ is close to 1

we branch on the left-hand side we assume that pairings i and i' are not covered together while the right-hand side assumes the two are covered by the same path. An example for when $\sum_{k:b_{1k}=b_{4k}=1} x_k$ is close to 1 is given by Figure 7.7. It is likely that pairings f_1 and f_4 are contained in the same roster-line. The branching strategy imposes subsequently that all columns that will be generated from then on either contain pairing f_1 and f_4 together or contain neither pairing.

Note that our pairing path incidence matrix B has non-zero columns since a zero column would mean that no pairing is included in a path which would not be included in the optimal solution. The columns in our pairing path incidence matrix B are distinct, since we only allow unique paths to enter the column B, because identical paths would never improve the objective function given that the cost vector is positive. In contrast to the theorem, we do not have the equality $Bx = \mathbb{1}$, but the inequality $Bx \geq \mathbb{1}$. However, we can easily work around this by taking subpaths $\overline{P} \subset P$ and adjust the path incidence matrix B accordingly such that we obtain the equality $Bx = \mathbb{1}$. Of course subpaths of paths are still feasible paths.

When we do not satisfy the fractional condition of the theorem, the solution found was already optimal. In this way the branching tree evolves in a more balanced way since branching in both direction eliminates many variables from the process.

7.4.3. Initial Matrix

There are multiple choices for initial matrix B'. The most trivial choice is the solution where every crew member executes exactly one pairing. Then $B' = I_{|V|}$, the identity matrix with length the number of pairings available. While this choice is obviously a feasible solution, one already start with as many columns as pairings that should be rostered. Therefore the initial column generation algorithm has to start with an already vast matrix which may slow down the process. It might be beneficial to start with a small subset of columns that can already produce a feasible solution, which is better than the trivial choice. We use a simple greedy algorithm to construct such a subset of

columns given by Algorithm 2.

```
Data: Pairings
Result: Feasible initial matrix
initialization;
for Pairing do
    Check if crew member is available;
    if Crew member available then
        Assign pairing to crew which ended last pairing the latest;
    else
        Create new crew member;
        Assign pairing to new crew;
    end
    Update available crew
end
```

Algorithm 2: Greedy Algorithm

This algorithm can provide a limited subset of columns that can produce a feasible integer solution. We call this initial matrix B_{gr}. Using initial matrix B_{gr} the column generation process can proceed faster since the linear program involved starts off with a very restricted number of variables compared to the trivial choice $B' = I_{|V|}$.

7.5. Bottleneck Indication

Even though the previous solutions shed some light on the efficiency of the flight schedule, in order to find out which flight schedule is the most efficient, several flight schedules should first be composed and should all be tested by the previous models for crew productivity. This process takes up a lot of time and since it only supplies us with some final numbers, it does not tell us why one schedule is better than another. Therefore, we introduce a new model similar to the models described in Section 7.3 that can provide us with some more insight with respect to the efficiency of the flight schedule and can help in improving the schedule beforehand. The idea is to use the solution of linear program (7.4), the minimum required crew members to realize a certain flight schedule. We call this solution C_{sol}. Then we find out what pairings we execute when we do not have C_{sol} but $C_{sol} - x$, x crew members fewer than necessary to execute the entire flight schedule. We use the same digraph as in Section 7.3, but we introduce a couple changes:

- The number of crew nodes are modified to $C_{sol} - x$ and the arcs going to and from these nodes are adjusted accordingly.
- For all $a \in A$, we define the cost of arc a as $c_a = 0$ **except** when $a = (f_1^i, f_2^i)$ for all $f \in F$ and $0 \leq i \leq 4$.
- For arcs of the form (f_1^i, f_2^i) for all $f \in F$ and $0 \leq i \leq 4$, we modify the arc cost to be the total *block time* of pairing $f \in F$.

The total block time of a pairing is the amount of time the aircraft is in the air in the corresponding pairing. Then we formulate the linear program as follows:

$$
\begin{aligned}
& & \max \sum_{a \in A} & c_a x_a & & \\
\text{subject to} & & x_a & \leq & u_{max}(a) & \quad \forall a \in A \\
& & x_a & \geq & u_{min}(a) & \quad \forall a \in A \\
& \sum_{a \in \delta^+(v)} x_a - \sum_{a \in \delta^-(v)} & x_a & = & s_v & \quad \forall v \in V \\
& & \sum_{i=1}^{4} \sum_{f \in F_i} & x_{f_1^i f_2^i} & \leq 1 & \quad \forall f \in F.
\end{aligned}
\tag{7.7}
$$

Knowing that there are not enough crew members to perform the entire schedule, this model identifies the pairings that are executed when we maximize over the total amount of block time of the schedule. But even more interesting are the pairings that are not executed. These pairings are the bottleneck pairings in the sense that they are the least efficient pairings when it comes down to crew productivity with respect to block time.

8. Results

In this chapter we discuss the results of the solution methods of the previous chapters. First the results of the pairing model and the retiming pairing model of Chapter 5 based on the different flight days in July and June 2019 are presented. Then, several key performance indicators regarding crew productivity based on the models of Chapter 7 for different flight schedules are discussed where we change the length of the flight schedule, its composition of flights and the pairing method of Chapter 5 that is used. The network flow models of Section 7.3 and Section 7.5 as well as the column generation model of Section 7.4 are tested.

8.1. Pairing and Retiming

We use the solution methods of Chapter 5 to determine the optimal assignment of double pairings. First the weighted set cover approach of Section 5.2 is used to determine how many pairings are needed in total and which link duty periods should be grouped together to form double pairings per day in the summer period when not using the crew linking document. In Table 8.1 the total number of single pairings, the number of double pairings and the total number of pairings are presented. To compare the results with the traditional pairings method of Section 5.1, the number of double pairings in the crew linking document is also included. Because the primary goal is to minimize the total length of the maximum number of double pairings, we use cost $c_P = \frac{d_P}{d_{max}}$ for all double pairings P where d_P corresponds with the length of pairing P and $c_P = 1$ for all single pairings P. We have set the maximum allowed work time $d_{max} = 12.5$ hours for all double pairings, since longer pairings are not advisable taking into account crew fatigue. Comparing with the results for cost function $c_P = 1$ for all pairings P tells us that Table 8.1 yields the maximum number of double pairings. For a destination specification of the double pairings, see Appendix A.4.

Day	Total Single Pairings	Total Double Pairings	Total Pairings	Crewlinking Double Pairings
Day 1	107	9	98	7
Day 2	99	8	91	4
Day 3	102	6	96	3
Day 4	104	10	94	7
Day 5	106	8	98	6
Day 6	108	7	101	6
Day 7	107	13	94	9

Table 8.1: Results of the weighted set cover approach

In Table 8.2 the results are presented for the maximum weight maximum cardinality retiming model created in Section 5.3. That is, when we create three duplicates of the same link duty period, one 10 minutes earlier and one 10 minutes later and one original. Because we maximize (instead of minimize) the weight in this model, we choose weight $w_P = \frac{d_{max}}{d_P}$ for every double pairing P and $w_P = 1$ for every single pairing P. Artificial high weights $w = 1000$ are chosen for all arcs between the same pairing nodes to ensure every pairing is matched at most once. For a full overview including routes, see Appendix A.5.

Day	Total Single Pairings	Total Double Pairings	Total Pairings
Day 1	107	9	98
Day 2	99	9	90
Day 3	102	6	96
Day 4	104	11	93
Day 5	106	11	95
Day 6	108	8	100
Day 7	107	14	93

Table 8.2: Results of the retiming approach

8.2. Network Flow Models

First we run the network flow models of Section 7.3 for the original schedule (S) from Monday 15-07-2019 up to and including Sunday 28-07-2019 and we start with an initial maximum number of crew members of 140. The total blocktime of the entire schedule is 5840 hours and the schedule contains 861 pairings where the construction method is based on the crew linking document of Section 5.1. The pairings considered are limited to pairings based in Amsterdam. The operational slack time between pairings is currently set on 80 minutes. We test the schedule for the extended network flow models proposed in Subsection 7.3.2 and Subsection 7.3.3.

In order to find out the impact of different flight schedules, especially regarding night pairings, on the crew productivity, we modify the original schedule (S) in several ways. First we create a schedule where all the pairings are one hour later (S+1). Similarly, we make a schedule where all the pairings are one hour earlier (S-1). Also, we include a schedule where all the pairings that generate recuperation periods are modified such that they do not generate recuperation time (SNR). That is, a pairing ending at 03:40 is pushed back 40 minutes in this schedule. We include an alternative schedule for when the aircraft spare capacity blocks are split up (SOP) and a hypothetical schedule for when another aircraft is used to withdraw flights from the night time hours (SAC). Additionally, we include the schedules based on the set covering method in Section 5.2

(SCC) and a schedule based on the retiming model of Section 5.3 (SRT). In order to check whether the rosters can be extended for more than the 14 days considered, we run the model twice for flight schedules spanning 28 days; once for schedule (SS1), from Monday 01-07-2019 up to and including Sunday 28-07-2019 and once for schedule (SS2), from Thursday 04-07-2019 up to and including Wednesday 31-07-2019. These are extended versions of the original schedule. The results are expressed in minimum number of crew members required, average block time[1] per crew member and total CPU time in Table 8.3:

Schedule	Minimum Crew	Average Blocktime (h)	CPU Time (s)
S	115	51h	1998s
S+1	118	49.6h	2552
S-1	119	49.2h	2246s
SNR	114	51.4h	2340s
SOP	114	51.4h	1910s
SAC	113	51.7h	1965s
SSC	113	51.7h	1983s
SRT	112	52h	2140s
SS1	115	102h	9882s
SS2	115	102h	9882s

Table 8.3: Results of the model proposed in Subsection 7.3.2 of various schedules

Because the results for the schedules SOP and SNR are similar, it is worth computing the total idle time and the total amount of FTE required for both of these schedules using the model proposed in Subsection 7.3.3. We also show the computational results of the corresponding model in the Table 8.4. Note that the model of Subsection 7.3.3 actually creates unpersonalized rosters. A gantt chart of one of the rosters entirely is attached in Appendix A.1.

Schedule	Total Idle Time (d)	Total FTE Time	CPU Time (s)
SOP	647.7	1356.7/1596	3263s
SNR	637.2	1346.2/1596	2786s
SAC	632.9	1345.2/1582	3243s

Table 8.4: Results of total idle time and total FTE for various schedules

Finally, the results of the bottleneck model of Section 7.5 are presented. We test the model on the original schedule (S) and use 114 crew nodes, one fewer than is required

[1]This exceeds the maximum allowed blocktime, because this is a optimal crew rostering model and thus only contains pairing duties that generate block time.

to perform the entire schedule. In Table 8.5 the pairings are presented that would be eliminated when maximizing the total block time for all pairings.

Pairing-Route	Check-in time	Check-out time	Block time
AMS-OLB-AMS	2019-07-22 04:35:00	2019-07-22 11:20:00	280
AMS-GRO-AMS	2019-07-22 11:45:00	2019-07-22 18:00:00	260
AMS-PSA-AMS	2019-07-22 19:40:00	2019-07-23 01:55:00	250
AMS-NCE-AMS	2019-07-24 05:35:00	2019-07-24 11:45:00	240
AMS-INN-AMS	2019-07-24 12:40:00	2019-07-24 18:15:00	190
AMS-LJU-AMS-VLC VLC-AMS-VLC-AMS	2019-07-21 05:35:00	2019-07-23 01:00:00	790

Table 8.5: Eliminated pairings when maximizing for total block time

8.3. Column Generation

We share the results of the column generation approach discussed in Section 7.4. In order to minimize the total number of required crew members, we attach positive penalty cost $c_a = 1$ for arcs of the form (s, f) for all $f \in F$ and all other arcs have cost 0. The model is run on a subset of the original schedule $S' \subset S$ where S' runs from Monday 15-07-2019 up to and including Sunday 28-07-2019 and only contains 25 pairings a day. The minimum number of crew members necessary is determined in advance by the model created in Subsection 7.2.1 and equals 32. We start with two different initial matrices, the trivial variant $B' = I_{|V|}$ and the greedy variant $B' = B_{gr}$ both discussed in Subsection 7.4.3. In this case B_{gr} has 62 columns. The column generation approach is embedded in a branch and bound structure based on the branching strategy discussed in Subsection 7.4.2. We apply the branching strategy on the rows i, i' for which

$$\sum_{k:b_{ik}=b_{i'k}=1} x_k$$

is closest to 1 and branch on the 1-tree. In order to speed up the process, we branch on the three row combinations (if they exist) for which the above summation is closest to 1. Additionally, we branch on every row combination i, i' for which

$$\sum_{k:b_{ik}=b_{i'k}=1} x_k > 0.9.$$

After the branching strategy, all previous generated columns are discarded and we start with a subset of columns depending on the initial matrix. When we consider the initial

matrix $B' = I_{|V|}$, we keep the matrix that corresponds with all pairings taken separately *except* for those pairings we branch upon. They are grouped together in a column. After the branching procedure when using the greedy initial matrix $B' = B_{gr}$, we keep all the variables of the original initial matrix $B' = B_{gr}$ *except* those who have become unfeasible by the branching. These pairings are replaced by the unit vectors that take these pairings separately. In this way we always keep a feasible solution at the beginning of the column generation process.

We keep track of the number of columns generated, the upper bound, which is determined by the number of non-zero variables, and the number of row combinations we branch upon, the *branchpaths* in that node. In Table 8.6 the results are shown for the matrix $B' = B_{gr}$.

Nodes	Columns	Upper bound	Branchpaths
1	167	62	3
2	197	137	3
3	211	141	3
4	204	151	3
5	167	145	3
6	170	136	3
7	106	102	3
8	91	88	3
9	91	88	3
10	91	91	4
11	60	59	3
12	68	79	3
13	64	71	5
14	56	62	9
15	63	59	6
16	54	47	3
17	52	55	3
18	45	49	3
19	51	47	3
20	42	40	3
21	44	37	3
22	39	32	NaN

Table 8.6: Results of the column generation approach with initial matrix $B' = B_{gr}$

Sometimes there will be branched in such a way that the initial matrix $B' = B_{gr}$ does not provide a feasible solution anymore. By keeping a feasible solution after the branching, the upper bound can increase after the node exploration. Also, in the table we only give the number of columns that are generated and not columns in the initial matrix. Therefore the upper bound can be a greater than the column value in Table 8.6.

The results for initial matrix $B' = I_{|V|}$ are shown in Table 8.7.

Nodes	Columns	Upper bound	Branchpaths
1	152	125	3
2	141	117	3
3	170	123	3
4	167	126	3
5	166	119	3
6	208	121	2
7	197	114	3
8	181	113	3
9	161	108	3
10	144	113	4
11	148	102	1
12	164	101	3
13	195	103	3
14	144	98	9
15	155	94	6
16	163	96	3
17	165	96	3
18	142	100	3
19	104	86	3
20	144	82	3
21	126	74	3
22	121	73	3
23	102	60	2
24	101	72	3
25	96	53	3
26	101	51	3
27	86	54	3
28	97	64	1
29	101	50	3
30	80	40	3

Table 8.7: Results of column generation approach with initial matrix $B = I_{|V|}$

The computation time per node ranges from 200 to 600 seconds depending on the number of columns being generated. That means that the entire process can take up more than 3 hours. Because the computation time would increase even more when we apply this approach to the entire flight schedule including free time periods, further investigation of this solution method is not exploited.

9. Conclusion

In this chapter we first address the initial research questions we formulated in the introduction. Then, the practical results as well as the mathematical performance of each of the models individually are discussed, but these results are also compared to the results and performance of the different models. Furthermore, recommendations for further research based on these findings are presented.

9.1. General Conclusion

We return to the two initial research question we have phrased in the introduction:

> *What is the exact impact of various sorts of night flights concerning the additional labor requirements of night flights compared to day flights and how do these requirements affect the productivity of the cockpit crew?*

The first part of this question has been answered thoroughly in Chapter 4, where the labor requirements are discussed for various flights with the emphasis on the requirements and restrictions for variations of flight time arrivals and departures. Chapter 6 has illuminated that the second part of the question cannot be answered considering night flights alone, but should be considered within the context of the complete flight schedule. Finally, Chapter 5 and Chapter 7 provide solution methods as to compute the crew productivity. The impact of these night flights can be found by comparing the results of various schedules in Table 8.3, created by the model of Subsection 7.3.2. The second research question we formulated was:

> *What are the exact costs we make when utilizing an extra aircraft and exact costs we save on cockpit crew when we use that aircraft to replace night flights with day flights?*

Although this thesis has focused mainly on the crew productivity based on various schedules, this research question can be answered with Table 8.3 and with the consultation of the finance department to complete the missing cost values. In Table 8.3 we find that the flight schedule with an extra aircraft (SAC) requires two fewer crew members than the original schedule (S). This means two net first captains and two net first officers. The costs of a captain and first officer together is €300.000. Including the tax gross up multiplier 1.4, this equals €420.000. For the two extra captains and two extra first officers in total, €840.000 is calculated. Comparing this with the total cost of an extra aircraft during the summer period of €3.000.000, we conclude that it is not lucrative to use an extra aircraft to improve the productivity of the cockpit crew by the reduction of night flights.

9.2. Network Flow Models

The results for the network flow model discussed in Table 8.3 show promising results for various flight schedules. While the use of an extra aircraft (SAC) can result in the need of two net crew members fewer compared to the original schedule, splitting up the spare capacity blocks of an aircraft to withdraw pairings from the night time hours, can already save one net crew member (SOP). In general we see that the number of night pairings affects the number of necessary crew members exponentially: the removal of the first couple night pairings leads to a higher reduction of necessary crew members than the removal of night pairings thereafter.

Concerning total idle time and total FTE, we see that the scenario with the spare aircraft capacity split up (SOP) yields worse results than the scenario where all the recuperation generating pairings have been eliminated (SNR). This means that even though both schedules can be performed by the same number of crew members, the (SOP) requires more crew time, because the optimal roster has more idle time between the pairings.

The bottleneck network flow model gives an idea of what pairings are the least productive when it comes down to maximizing block time hours. Most pairings that are eliminated are pairings that are relatively short. For these pairings the block time relative to the check-in time the next day minus the check-in time of that pairing considered is relatively small, because every pairing has the same rest period. When these pairings are also night pairings or have early check-in times, the restrictive labor requirements make these pairings even more likely to be eliminated.

Even though the lay-over pairings include a lot of block time, these pairings span two working days where the check-in on the first day is early in the morning while the check-out of the next day is late at night. This is because the time between the two duty periods in this pairing includes 24 hours while the mandatory rest requirement is only 13 hours, see Figure 5.1. As a consequence these pairings are less productive.

The computational performance for all network flow models is relatively good with the highest computation times not exceeding two hours. The computation times of the bottleneck model are omitted, because these are similar to these of model 7.3.3. In general we experience a higher computation time for model 7.3.3 than for model 7.3.2.

Testing on extended flight schedules (SS1 and SS2) yields the same number of minimum number of crew members required to perform the schedule, meaning that the rosters are extendable. The main drawback of the network flow models is that we cannot set requirements on individual roster-lines such as the maximum allowed total work time or total number of working days. Also, the structure of the graph always imposes the max 4 days of work and min 3 days of free time pattern, but in practice one can deviate from this pattern as long as the 20 working days in 35 roster days will not be exceeded.

9.3. Column Generation

In contrast to the network flow models, the column generation approach has the potential of being much more versatile. The reason for this is that one can control what columns enter the matrix and which columns cannot enter the matrix. However, the column generation approach yields very poor computational results with computation times exceeding 3 hours. This is because the columns that are generated give rise to a very fractional solution in the sense that many columns are used to solve the restricted master problem. This leads to a high upper bound of the solution and the many fractional variables cause the branching strategy to be inefficient and sometimes inadequate.

Remarkable is that the choice of the initial matrix has a large effect on the column generation process. We see that the use of the initial matrix $B' = B_{gr}$ leads to fewer node explorations and eventually leads to a better solution than initial matrix $B' = I_{|V|}$. The branching strategy works better when there are many row combinations with $\sum_{k:b_{ik}=b_{i'k}=1} x_k$ close to 1, which is more likely to happen when the initial matrix contains a lot of columns that take multiple pairings together, such as for initial matrix $B' = B_{gr}$. We observe that if we branch on the rows for which $\sum_{k:b_{ik}=b_{i'k}=1} x_k$ has the greatest value for the initial matrix $B' = I_{|V|}$, we sometimes branch on rows that cannot produce an optimal solution. Hence the final solution provided with the choice of initial matrix $B' = I_{|V|}$ is suboptimal.

9.4. Pairing Models

The two pairing methods provided in Section 5.2 and Section 5.3 can yield significantly more double pairings than the classical pairing creation method of Section 5.1. This creation of pairings leads to better results in the network flow model in Table 8.3, meaning that both pairing methods suggested can increase the crew productivity. The retiming model in Section 5.3 can create a set of pairings that yield even better crew productivity than the set covering model in Section 5.2. The main reason for the improvement of both methods is that in these pairing methods, the total number of pairings decreases, while maintaining the same number of block hours. Consequently, the number of mandatory rest hours decreases and hence the crew can work more productively. An important assumption is that we do not have a limit on total block time of individual roster-lines. As crew becomes more productive, sometimes individual crew members exceed their maximum allowed working time in a certain roster period. One should also keep in mind that the minimum cost flow models only take the planned schedule into account. In the actual planning stages, double pairings are more sensitive to delay than single pairings, since a delay in the first link duty period can lead to a consequent delay of the second link duty period.

9.5. Recommendations

The main drawback of the network flow models for the calculation of crew productivity is that we cannot set requirements on individual roster-lines. Knowing that the maximum total number of block hours per crew member can be a limitation, it is worth exploring methods that can account for this restriction. The column generation method does allow limitations set on individual roster-lines, but yields very poor computational results. However, because we see a significant better result for the column generation when using an initial matrix that contains a decent feasible solution, one can try using column generation with the solution provided by the minimum cost flow model of Subsection 7.3.2. as initial matrix.

Another limitation on this model is the max 4 days work, 3 days of free time structure of the roster. This structure is very common, because it guarantees that no more than 8 of the 14 days can be working days, corresponding to the maximum 20 working days in the 35 day roster period. However, sometimes the actual roster-lines deviate from this pattern. The model would be more exact when allowing different roster-line structures as long as no more than 8 of the 14 days can be working days. This is also possible with a more refined column generation approach since the columns entering the matrix can be filtered for number of working days.

Finally, this thesis assumes that every recuperation generating pairing generates a recuperation period. This assumption is justified by observing that the structure of working days considered leaves very little room to avoid recuperation periods in contrast to the 'tricks' of the gross schedule described in Chapter 5 and even if some recuperation periods are avoided, early computations show that the difference would be negligible for the schedules considered. When allowing different structures of working days and free time periods, more options arise to avoid recuperation periods and one should find a more refined method as to when to include recuperation periods.

Appendices

A. Appendix

A.1. Unpersonalized roster example

Figure A.1 is an illustration of a complete crew roster for flight schedule SAC produced by model of Subsection 7.3.3. Normal pairings are indicated by green, night pairings by red and recuperation generating pairings by purple. The pairing times indicated represent the check-in time and possible next check-in time on the left and right-hand side respectively. The blue bar represents the free time period which may overlap with part of the pairing.

A.2. Taxi appendix

Route	Total taxi time (h)
AMS-RTM	01:00
AMS-EIN	01:40
AMS-GRQ	02:15
RTM-EIN	01:30
RTM-GRQ	03:00
EIN-GRQ	03:00

A.3. Conversion Leg set to Pairings

Figure A.2 describes the entire process of a leg set being converted to two separate pairings, with in between the duty periods created by Algorithm 1.

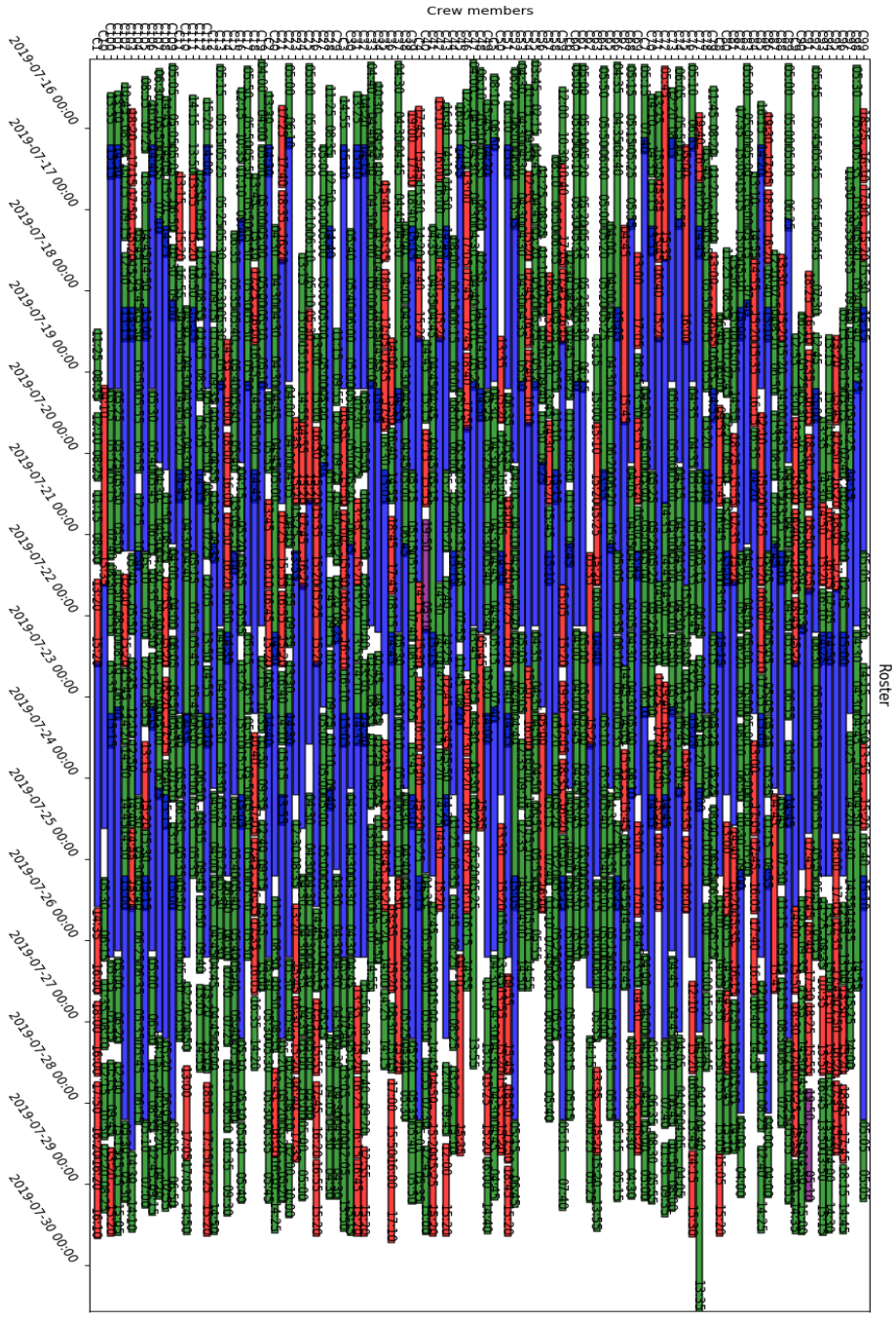

Figure A.1: Gantt chart representing unpersonalized rosters for schedule SAC

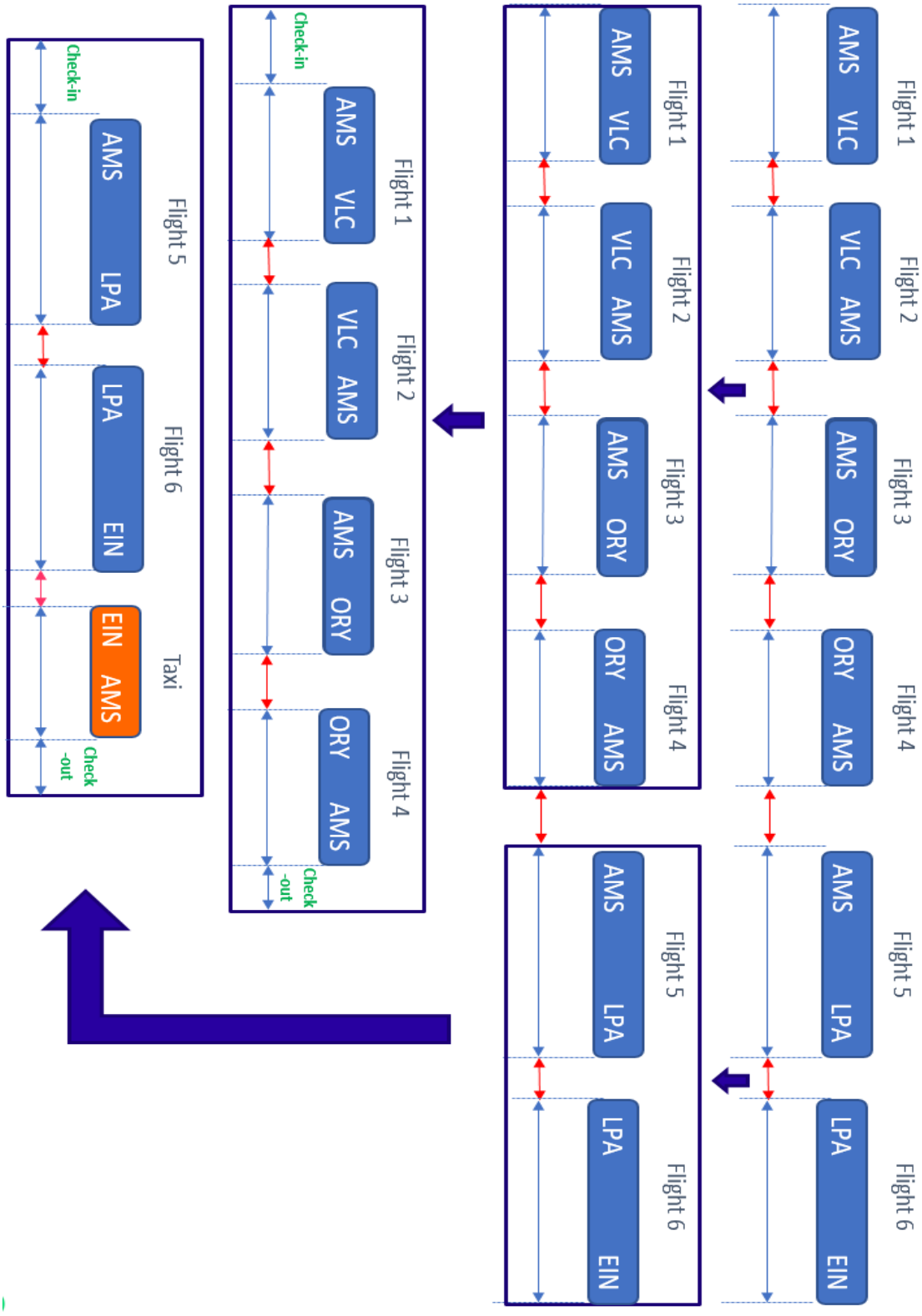

Figure A.2: Conversion of legs to duty periods and eventually to pairings

A.4. Pairing appendix

Day 1
-RTM-IBZ-RTM-PUY-RTM
-EIN-PRG-EIN-VLC-RTM
-RTM-ALC-RTM-EGC-RTM
-RTM-BCN-EIN-NCE-EIN
-EIN-CPH-EIN-KRK-EIN
-RTM-PMI-RTM-MPL-RTM
-EIN-BCN-RTM-VIE-RTM
-AMS-VRN-AMS-NCE-AMS
-AMS-GRO-RTM-GRO-AMS

Day 2
AMS-ALC-RTM-EGC-RTM
RTM-IBZ-RTM-NCE-RTM
RTM-SPU-RTM-GRO-RTM
EIN-NCE-EIN-IBZ-EIN
AMS-NCE-AMS-KTW-AMS
RTM-GRO-RTM-TLN-RTM
AMS-BCN-EIN-RJK-EIN
AMS-PSA-AMS-GRO-AMS

Day 3
AMS-VRN-AMS-HEL-AMS
RTM-PMI-RTM-GRO-RTM
EIN-NCE-EIN-BCN-AMS
RTM-ALC-RTM-EGC-RTM
AMS-NCE-AMS-GRO-AMS
AMS-IBZ-RTM-MPL-RTM

Day 4
AMS-ALC-RTM-EGC-RTM
AMS-VRN-AMS-PSA-AMS
AMS-VRN-AMS-NCE-AMS
RTM-ALC-AMS-ORY-AMS
RTM-TLN-RTM-SPU-RTM
EIN-KRK-EIN-VLC-EIN
EIN-BLQ-EIN-NCE-EIN
EIN-BCN-EIN-CPH-EIN
EIN-RJK-EIN-PRG-EIN
RTM-FCO-RTM-GRO-RTM

Day 5
AMS-NCE-AMS-GRO-AMS
AMS-OLB-AMS-KTW-AMS
EIN-PRG-EIN-BCN-EIN
RTM-MPL-RTM-IBZ-RTM
RTM-PSA-RTM-VIE-RTM
RTM-GRO-RTM-BCN-RTM
EIN-CPH-EIN-NCE-EIN
AMS-VRN-AMS-PSA-AMS

Day 6
RTM-EGC-RTM-AGP-RTM
RTM-BCN-RTM-PSA-RTM
RTM-GRO-RTM-SPU-RTM
EIN-BLQ-EIN-BCN-RTM
RTM-ZAD-RTM-PUY-RTM
AMS-NCE-EIN-NCE-AMS
AMS-VRN-AMS-GRO-AMS

Day 7
AMS-GRO-AMS-VRN-AMS
EIN-VLC-EIN-PRG-AMS
AMS-NAP-AMS-ORY-AMS
AMS-NCE-AMS-GRO-AMS
AMS-PMI-RTM-MPL-RTM
AMS-VRN-AMS-PSA-AMS
EIN-NCE-EIN-NCE-EIN
RTM-TLN-RTM-BCN-RTM
AMS-INN-AMS-VLC-GRQ
EIN-VLC-EIN-RJK-EIN
RTM-SPU-RTM-GRO-RTM
RTM-DBV-RTM-EGC-RTM
AMS-LJU-AMS-PSA-AMS

A.5. Retiming appendix

We refer to a link duty period starting at airport x shifted 10 minutes forward as x^+ and shifted 10 minutes backward as x^-. A link duty period unshifted at airport x is simply denoted by x.

Day 1
EIN-CPH-EIN-KRK-EIN
EIN-BCN-RTM-VIE-RTM
EIN-PRG-EIN-ARN-EIN
RTM-PMI-RTM-MPL$^-$-RTM$^-$
EIN$^+$-NCE$^+$-EIN-BLQ$^-$-EIN$^-$
EIN-VRN-EIN-NCE-EIN
RTM-EGC-RTM-PSA-RTM
RTM-IBZ-RTM-PUY-RTM
AMS-GRO-RTM-GRO-AMS

Day 2
AMS-NCE-AMS-KTW$^-$-AMS$^-$
AMS-PSA-AMS-GRO$^-$-AMS$^-$
AMS-BEG-AMS-VRN$^-$-AMS$^-$
RTM-SPU-RTM-GRO-RTM
RTM-NCE-RTM-IBZ-RTM
RTM$^+$-PUY$^+$-RTM-EGC$^-$-RTM$^-$
EIN-NCE-EIN-IBZ-EIN
RTM-GRO-RTM-TLN$^-$-RTM$^-$
AMS-BCN-EIN-RJK$^-$-EIN$^-$

Day 3
AMS-INN-AMS-HEL$^-$-AMS$^-$
RTM-PMI-RTM-GRO-RTM
RTM-PSA-RTM-MPL$^+$-RTM$^+$
EIN-NCE-EIN-BCN-AMS
RTM-SPU-RTM-EGC-RTM
AMS-GRO-AMS-NCE$^+$-AMS$^+$

Day 4
RTM-GRO-RTM-PUY-RTM
AMS-VRN-AMS-BCN$^-$-AMS$^-$
AMS-VRN-AMS-NCE$^-$-AMS$^-$
RTM$^+$-PMI$^+$-AMS-ORY$^-$-AMS$^-$
RTM-NCE-RTM-TLN-RTM
EIN-BCN-EIN-CPH-EIN
EIN-BLQ-EIN-NCE-EIN
EIN-PMI-EIN-PRG-EIN
EIN-RJK-EIN-KRK-EIN
RTM$^+$-EGC$^+$-RTM-SPU$^-$-RTM$^-$
AMS$^+$-LJU$^+$-AMS-PSA$^-$-AMS$^-$

Day 5
AMS$^+$-GRO$^+$-AMS-PSA$^-$-AMS$^-$
AMS$^+$-VRN$^+$-AMS-PSA$^-$-AMS$^-$
AMS$^+$-OLB$^+$-AMS-KTW$^-$-AMS$^-$
RTM$^+$-PUY$^+$-RTM-PMI$^-$-RTM$^-$
EIN-PRG-EIN-BCN-EIN
RTM-MPL-RTM-IBZ-RTM
RTM-DBV-RTM-VIE$^+$-RTM$^+$
EIN-ARN-EIN-CPH$^+$-EIN$^+$
RTM-GRO-RTM-PSA$^+$-RTM$^+$
RTM-SPU-RTM-GRO$^+$-RTM$^+$
AMS$^+$-NCE$^+$-AMS-GRO$^-$-AMS

Day 6
RTM-NCE-RTM-EGC-RTM
RTM-TLN-RTM-BCN$^+$-RTM$^+$
EIN-BCN-EIN-BLQ-EIN
RTM-GRO-RTM-SPU-RTM
AMS$^+$S-GRO$^+$-AMS-PSA$^-$AMS$^-$
AMS$^+$-VRN$^+$-AMS-PMI$^-$-AMS$^-$
AMS-NCE-EIN-NCE-AMS

Day 7
EIN-RJK-EIN-BCN-AMS
AMS-BCN-AMS-VRN$^-$-AMS$^-$
AMS$^+$-GRO$^+$-AMS-GRO$^-$-AMS$^-$
RTM-SPU-RTM-PUY-RTM
AMS$^+$-PSA$^+$-AMS-PSA$^-$-AMS$^-$
AMS-NAP-AMS-ORY-AMS
EIN-NCE-EIN-NCE-EIN
AMS$^+$-INN$^+$-AMS-VLC$^-$-AMS$^-$
EIN-VLC-EIN-PRG-EIN
RTM$^+$-SPU$^+$-RTM-PUY$^-$-RTM$^-$
RTM-TLN-RTM-MPL-RTM
RTM-BCN-RTM-GRO$^+$-AMS$^+$
AMS$^+$-OLB$^+$-AMS-VRN$^-$-AMS$^-$
AMS$^+$-LJU$^+$-AMS-PSA$^-$-AMS$^-$
AMS-NCE-AMS-PSA-AMS

Bibliography

[1] Barnhart, C., Cohn, A. M., Johnson, E. L., Klabjan, D., Nemhauser, G. L., and Vance, P. H. (2003). Airline crew scheduling. In *Handbook of transportation science* (pp. 517-560). Springer, Boston, MA.

[2] Borndörfer, R., Schelten, U., Schlechte, T., and Weider, S. (2006). A column generation approach to airline crew scheduling. In *Operations Research Proceedings 2005* (pp. 343-348). Springer, Berlin, Heidelberg.

[3] Cadarso Morga, L., and Marín Gracia, A. (2011). Integrated robust airline schedule development. *Procedia Social and Behavioral Sciences*, 20, 1041-1050.

[4] Conforti, M., Cornuejols, G., and Zambelli, M. G. (2014). Integer Programming.

[5] Cook, W. J., Cunningham, W. H., Pulleyblank, W. R., and Schrijver, A. (1997). Combinatorial Optimization.

[6] Deveci, M. and Demirel, N. Ç. (2018). Evolutionary algorithms for solving the airline crew pairing problem. *Computers Industrial Engineering, 115*, 389-406.

[7] Eltoukhy, A. E., Chan, F. T., and Chung, S. H. (2017). Airline schedule planning: a review and future directions. *Industrial Management & Data Systems, 117(6)*, 1201-1243.

[8] Guo, Y., Mellouli, T., Suhl, L., and Thiel, M. P. (2006). A partially integrated airline crew scheduling approach with time-dependent crew capacities and multiple home bases. *European Journal of Operational Research, 171(3)*, 1169-1181.

[9] Klabjan, D., Johnson, E. L., Nemhauser, G. L., Gelman, E., and Ramaswamy, S. (2001). Airline crew scheduling with regularity. *Transportation science, 35(4)*, 359-374.

[10] Klabjan, D., Johnson, E. L., Nemhauser, G. L., Gelman, E., and Ramaswamy, S. (2002). Airline crew scheduling with time windows and plane-count constraints. *Transportation science, 36(3)*, 337-348.

[11] Lusby, R., Dohn, A., Range, T. M., and Larsen, J. (2012). A column generation-based heuristic for rostering with work patterns. *Journal of the Operational Research Society, 63(2)*, 261-277.

[12] Maenhout, B. and Vanhoucke, M. (2010). A hybrid scatter search heuristic for personalized crew rostering in the airline industry. *European Journal of Operational Research, 206(1)*, 155-167.

[13] Mercier, A. and Soumis, F. (2007). An integrated aircraft routing, crew scheduling and flight retiming model. *Computers and Operations Research, 34(8)*, 2251-2265.

[14] Regeling werk- en rusttijden luchtvaart. (2008, 19 July). URL: `https://wetten.overheid.nl/BWBR0024175/2008-07-19/0.`

[15] Ryan, D.M. and Foster B.A. (1981). An integer programming approach to crew scheduling.

[16] Şahin, G. and Yüceoğlu, B. (2011). Tactical crew planning in railways. *Transportation Research Part E: Logistics and Transportation Review, 47(6)*, 1221-1243.

[17] Transavia. (September 2018). Crew-linking document of summer 2019.

[18] Transavia. (July 2017). Collectieve arbeidsovereenkomst Transavia Vliegers.

[19] Yaghini, M., Khoshraftar, M. M., and Fallahi, M. (2013). A hybrid algorithm for artificial neural network training. *Engineering Applications of Artificial Intelligence, 26(1)*, 293-301.

[20] Yunes, T. H., Moura, A. V., and De Souza, C. C. (2005). Hybrid column generation approaches for urban transit crew management problems. *Transportation Science, 39(2)*, 273-288.