A dynamic optimization model on the routing and maintenance scheduling of aircraft for individual tasks

A.S.A. Steenkamp

A dynamic optimization model on the routing and maintenance scheduling of aircraft for individual tasks

by

A.S.A. Steenkamp

Master Thesis

in partial fulfilment of the requirements for the degree of

Master of Science
in Mechanical Engineering

at the Department Maritime and Transport Technology of Faculty Mechanical, Maritime and Materials Engineering of Delft University of Technology
to be defended publicly on Thursday August 27, 2020 at 13:00.

Student number:	4315413	
MSc track:	Transport Engineering and Logistics	
Report number:	2020.MME.8445	
Thesis committee:	Prof. dr. R. Negenborn,	TU Delft, committee Chair, 3ME
	Dr. F. Schulte,	TU Delft, supervisor, 3ME
	Dr. L. Pahlavan,	TU Delft, 3ME
	Dr. M. Moeller,	TU Delft, EEMCS
Date:	August 27, 2020	

TU Delft

Preface

Before you lies the master thesis 'A dynamic optimization model on the routing and maintenance scheduling of aircraft for individual tasks'. This thesis was written to obtain the degree of Master of Science at the Delft University of Technology in Mechanical Engineering with a specialization in Transport Engineering & Logistics.

I got the inspiration for this thesis when doing an internship at Lufthansa Technik, where I worked on a project regarding optimizing the scheduling of line maintenance. During this time, I learned a lot about the current state of the aviation industry, which allowed me to not only consider the research questions from a scientific point of view but also from a practical one.

I could not have completed this thesis on my own. Firstly, I would like to thank my daily supervisor Frederik Schulte, who supported me throughout this period. His insight when discussing solution approaches and his unwavering optimism were instrumental to the completion of this work. Secondly, I would like to thank my step-mother, Professor Valarie Zeithaml, for giving me tips on how to efficiently perform a literature study and for always offering me a second pair of eyes on written work.

Lastly, I would like to thank my fellow students, who provided a feeling of comradery as they underwent similar but different challenges, my friends, who at times could offer a much needed break from working, and my parents, who provided me with emotional support.

A.S.A. Steenkamp
Delft, August 2020

Abstract

Increasing competition amongst airlines necessitates them to improve the efficiency of their operations. Even though maintenance, repair, and overhaul (MRO) represent a significant portion of an airline's operational costs, aircraft maintenance scheduling is often still a manual process, producing suboptimal solutions. Airlines typically operate by congregating the bulk of the required maintenance tasks in extensive checks, called letter checks (A, B, C, or D). Letter checks require the aircraft to be taken out of operations and result in many tasks being executed before they are due, leading to more required maintenance over the aircraft's lifetime. The purpose of this study is to develop a methodology that provides flight routes to aircraft and plans the maintenance tasks individually within these routes over a given planning horizon with the objective of maximizing the utilization of the total remaining flying time of the fleet. To achieve this, tasks are planned as late as possible on overlays at a maintenance station, while being given a due date and a remaining number of legal flight hours that can be flown before execution is mandatory. For this purpose, we develop a mixed integer programming (MIP) model based on a city-day network representation. Because the computational burden of exact methods becomes too hefty for increasing problem sizes, several matheuristics have been developed to provide good solutions in quick fashion. The presented matheuristics either decompose the problem by aircraft or into time periods. The former constructs the flight routes and maintenance schedules aircraft per aircraft while the latter constructs them simultaneously in a rolling horizon fashion. For the rolling horizon matheuristics, several forecasting strategies have been designed as well. In an experimental study, one of the selected rolling horizon matheuristics was able to remove the need for aircraft to be taken out of operations for an A-check (the most frequently occurring letter-check), potentially saving up to $ 7.2 million per aircraft over a time period of ten years. Furthermore, the lost flying time, incurred by planning maintenance tasks before they are due, was decreased by over 98%, resulting in a higher utilization of the task intervals and less required maintenance over the aircraft's lifetime. Finally, the dissection of the A-check into its individual tasks led to a more phased maintenance schedule by attenuating the peaks in workload for the mechanics workforce. Our presented approach can be used by mid-sized airlines to optimize their maintenance schedules through increasing aircraft availability and reducing maintenance costs over the aircraft's lifetime.

Contents

List of Figures

List of Tables

1 Introduction

Since the deregulation of the intra-European air transport market in 1997, competition among airlines has intensified significantly [28]. From that moment, any EU air carrier was allowed to operate from any EU country. Low-cost carriers, improvement of high-speed rail networks, and increased price transparency provided by the internet have all contributed to this increase of competition. In this day and age, passengers have a broader choice in terms of routing and pay a lower price. Airlines need to find ways to improve their operational efficiency in the current competitive market of air transport. Even though maintenance, repair, and overhaul (MRO) represent a significant portion of an airline's operational costs (at least 9% in 2018 excluding overhead costs [18]), the planning process is far from optimized. In the industry, it is often still a very manual process, producing suboptimal solutions.

While maintenance is required and performed in all industries to maximize system availability and prevent or reduce the adverse effects of failures [12], MRO holds an especially important place in the airline industry due to heavy regulations. There is a distinguishable list of maintenance tasks for each aircraft that are to be carried out before a specific due date, with a repetitive interval. Each task has a unique task code, a determined amount of necessary man-hours for completion, an account of the required types of mechanics, and much more practical information. If a particular maintenance task has not yet been performed after its due date, the aircraft loses its airworthiness. These strict regulations, imposed by national and international authorities, have made the operational maintenance planning a complex problem. On the bright side, strict regulations have contributed to air travel being the safest mode of transportation. In 2010 aviation outperformed highway, transit, and railroad transportation in terms of the total number of fatalities, accidents per 100 million passenger mile traveled, and fatalities per 100 million passenger mile traveled by some margin [29].

Currently, most airlines plan the bulk of their MRO activities in extensive checks, called letter-checks (A, B, C, or D), which are performed once per a long interval (see chapter 2). During a letter-check, the aircraft is taken out of operations, and many maintenance tasks are executed together. In between the intervals of letter-checks, aircraft still need occasional maintenance checks every couple of days for more frequently occurring tasks. The main advantage of using extensive letter-checks lies in the relative ease of the planning, as one only has to plan one event for a large number of tasks. The structure of the standard maintenance checks is elaborated upon in chapter 2.

1.1. Research Problem

The potential for improvement in the maintenance scheduling for aircraft has caused much work to be published on finding optimal solutions for various types of maintenance planning and routing problems. A literature survey on this research is presented in chapter 2. However, much of this research focuses on the routing of aircraft, with maintenance requirements given as the basic constraint that an aircraft should spend the night at a maintenance station once every *d* days for a generic check. Research limiting the maintenance requirements to the scheduling of a generic check once every *d* days could be useful in combination with the use of letter-checks since the bulk of the maintenance tasks would be grouped inside of the letter-checks. The generic check could then contain all the smaller and more frequently required maintenance tasks.

However, the use of letter-checks for maintenance planning has two significant downsides:

1. During a check, an aircraft is taken out of operations for one or more days, resulting in a decrease in aircraft availability and a loss of potential revenue.
2. By grouping maintenance tasks with different intervals together in an extensive check, many tasks are executed before they are due, resulting in more required maintenance over an aircraft's lifetime.

These severe downsides of letter-checks have given rise to a new maintenance philosophy, where extensive letter-checks are dissected, and the tasks are planned at an individual level. In this philosophy, every ground time on a maintenance station is seen as an opportunity for maintenance. Transitioning from a letter-check towards an individual task-oriented maintenance strategy could be very beneficial for airlines. In a case study performed by Senturk and Ozkol (2018) [38] for a Turkish airline, they state that the necessary ground time for maintenance for an A340-family type aircraft could potentially have been reduced from 87 days to 15 days over a ten-year period if an individual task-oriented maintenance planning had been used instead of the standard letter-checks. Depending on the aircraft's utilization level, a day of operations may represent between $75k and $120k of additional revenue [11]. In the current literature, there is not much work done on individual task-oriented maintenance planning models for airlines. Some research has been published on maintenance planning and -routing models that consider a wider range of maintenance requirements for aircraft instead of generic checks, but most have not taken the airline's overall scheduling framework into account. Because the scheduling of aircraft maintenance tasks and the routing of aircraft is an interrelated problem, the overall scheduling framework can not be ignored and is elaborated upon in this work in section 1.4 and section 2.2. The problem description and the current state of the airline industry and literature lead to the following problem statement for this research:

Within the academic literature concerning an airline's maintenance scheduling optimization, limited knowledge exists on the building of an individual task-based maintenance routing and scheduling model that fits inside an airline's overall scheduling framework.

1.2. Research Objective

This thesis is conducted with the objective of enriching the academic community by developing a model with the capability of optimizing the flight and corresponding maintenance schedules for aircraft on an individual task-based approach. Sub-objectives of this thesis include:

- Creating insight from literature in possible model formulations that could be suitable for this application
- Composing a model capable of solving the problem
- Developing a method to achieve good quality solutions in an acceptable time
- Evaluating the possible impact of an optimized individual task-based maintenance scheduling and routing model on the airline industry

1.3. Research Questions

Based on the above-described research problem and objective, the main research question is defined as:

How can the efficiency and quality of the maintenance scheduling process in the airline industry be improved through optimization?

In order to answer this question as completely as possible the following key questions need to be answered:

1. What is the current state of the airline industry in terms of maintenance scheduling?
2. What is the current state of the literature regarding aircraft maintenance scheduling optimization?
3. How should a working model be designed to optimize the individual task-based maintenance scheduling?
4. What kind of solution method is well suited to solve the individual task-based maintenance scheduling problem in a reasonable time?
5. What is the impact of an optimized individual task-based maintenance scheduling on the airline industry?

1.4. Research Scope

The scheduling of an airline's operations and resources is an extremely complex and interrelated problem. Because of this complexity, it is the current practice to break up the resource planning into several stages. The five stages in which the different scheduling problems in the airline industry are divided are flight scheduling, fleet assignment, aircraft maintenance routing, crew scheduling, and tail assignment [24]. The maintenance scheduling of aircraft is not a stand-alone problem. Therefore, it is essential to get a view of the overall scheduling operations that an airline faces. These operations will be discussed briefly in this section and are elaborated further upon in chapter 2.

In the flight scheduling stage, an airline decides which flights it will operate, and during the fleet assignment, it is determined what type of aircraft will operate each flight. The flight schedule is created approximately one year in advance [21]. During the aircraft maintenance routing (AMR), individual flights are combined to form routes. It has been given this name because planners try to create flight routes that are expected to be feasible for smaller and more frequently occurring maintenance tasks [27]. During the crew scheduling stage, crew members are assigned to these routes with the aim to minimize crew cost and maximize various other objectives, such as quality of life and crew satisfaction. The crew schedules are determined approximately one month before the day of operations [21]. Only a few weeks or even days (this varies strongly between airlines) before the day of operations, the tail assignment (TA) is solved, where the created routes are assigned to specific aircraft (often referred to as tail numbers) [21]. During the tail assignment, an aircraft's initial location and individual maintenance needs are the key issues, since the route that is assigned in this stage needs to be compliant with the aircraft's individual maintenance needs. Adjustments to the routes created in the AMR might be required because these considerations and possible schedule disruptions could result in an infeasible maintenance schedule.

Maintenance scheduling in the airline industry is intertwined with its routing. Airlines usually only have a limited number of maintenance stations, which means that ensuring that the aircraft is in the right position at the right time fundamental to the problem. When operating with letter-checks, this is usually not too difficult. However, when transitioning from a letter-check to an individual task-oriented maintenance strategy, in which every ground time at a maintenance station is seen as a maintenance opportunity, this becomes much more challenging.

Since the routes constructed in the AMR are built far in advance and are not tail number specific, the planning of individual maintenance tasks inside ground times should be done during the TA. That is why the scope of this thesis will focus on solving the TA, meaning the optimal distribution of routes over the aircraft and the planning of maintenance inside them in an optimal way.

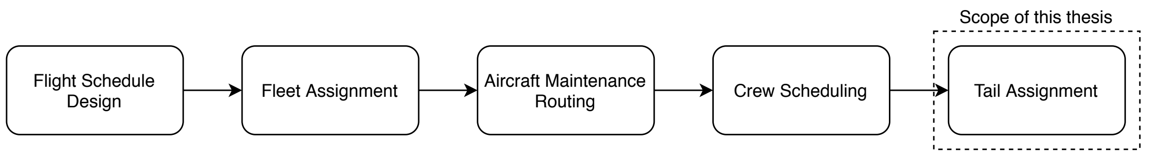


Figure 1.1: The scope of this thesis

1.5. Outline of the Thesis

In Table 1.1, an overview is given of the chapters in this thesis and the corresponding research questions. This table can be used to easily navigate throughout this thesis.

Table 1.1: Overview of the thesis

Chapter	Title	Research Questions
1	Introduction	1,2
2	Literature Review	1,2
3	Problem Definition	3
4	Solution Approach	4
5	Experimental Study	5
6	Impact on the Aircraft Maintenance Industry	5
7	Conclusion & Recommendations	-

2 Literature Review

Prior to this research, a literature study was conducted to investigate the current state and the main findings of the literature regarding airlines' maintenance scheduling operations. This chapter will summarize the main findings obtained during the literature search.

2.1. Basics of Aircraft Maintenance

Maintenance is the total of all activities performed throughout the life cycle of an item to retain it in a functioning state or to restore it to such a state [15]. Regular aircraft maintenance provides assurance of flight safety, reliability, and airworthiness through the performance of preventive and corrective maintenance. Preventive maintenance (referred to as scheduled maintenance in aviation) aims to retain the systems and components in a functioning state, whereas corrective maintenance (unscheduled maintenance) targets the restoration to this state after a failure has occurred. All the tasks that are to be performed to keep the aircraft in a continuous state of airworthiness are addressed in an airline's maintenance program. Due to the nature of our world, most systems and components will, at some point, deteriorate beyond a tolerable level or even fail completely [20].

2.1.1. History of Aircraft Maintenance

In the 1950s, aircraft carriers developed and proposed their own unique programs and processes for aircraft maintenance [39]. Senturk, Kavsaoğlu and Nikbay (2010) summarize of how this process evolved to one in which the authorities and industry work closer together [39]. In 1968 the Maintenance Steering Group 1 (MSG-1) was formed, which helped to develop the initial minimum scheduled maintenance recommendations for B-747 aircraft. The MSG-1 was a bottom-up approach where components—rather than the full system level–were the highest considered level [5]. In the decade hereafter, newly gained experience and insights led to the creation of the MSG-2 and eventually, the MSG-3, in which European and U.S. authorities, airlines, and manufacturers cooperated to update the procedures to apply to current and future aircraft. MSG-3 involves a top-down, consequence-of-failure-oriented approach. The failure analysis is conducted at the system level instead of the component level, as was the case in MSG-1 and MSG-2. Failure analysis at the system level means that the question is asked if the failure affects the safe operation of the aircraft itself [20]. The selected tasks in the MSG process are published in an officially approved document called the Maintenance Review Board Report (MRBR), which contains the initial minimum scheduled maintenance requirements for particular aircraft. The manufacturer of the aircraft publishes the Maintenance Planning Document (MPD), which provides extra information, support, and possible maintenance tasks to keep the aircraft in good condition throughout its lifetime. Using the MRBR and the MPD as a reference and adhering to their minimally set requirements, each aircraft carrier developed its own maintenance program, which needs to be approved by EU and national authorities.

Before the introduction of the Boeing 777, the tasks with a larger interval in the maintenance program were mandatorily split in the aforementioned letter-checks (A, B, C, and D). The typical A-check includes the visual inspection of the interior and exterior of the aircraft. C-checks include the inspection and functionality-testing of individual systems and components. During a D-check, the structurally important components are inspected, which requires the uncovering of the airframe, wings, and supporting structure [3]. The advantage of the letter-checks lies in the relative ease of the construction of the maintenance program. However, having lengthy checks during which an aircraft is halted on the ground is not the optimal strategy regarding the aircraft's availability. Furthermore, it results in distinctive peaks in the workload for maintenance personnel [3].

Since the second revision of MSG-3, coincident with the introduction of the Boeing 777, the use of letter-checks is no longer mandated for an airline's maintenance program. Maintenance tasks were not necessarily grouped into letter-checks and could be done at the most appropriate time for the equipment or system as long as the task interval was not exceeded. This makes the maintenance program more adaptable to the operator's needs [20].

The introduction of MSG-3 revision 2 has given rise to another maintenance program philosophy that views all the time an aircraft spends on the ground, as a result of their flight plan, as maintenance opportunities that could be utilized [38, 39]. The approach is to phase out the large letter-checks into smaller work packages. These work packages can then be fitted inside ground times on maintenance stations as a result of their flight plan, where night-layovers often are the best fitting candidates. Maintenance activities during which an aircraft is still in operation are known as line maintenance. A typical example of line maintenance is checking the landing gear for wear [20]. Maintenance activities that require an aircraft to be temporarily removed from operations are referred to as base maintenance. Two typical examples of base maintenance are the painting of the aircraft and engine removal or installation [20].

2.1.2. Different Types of Checks

In the book Aviation Maintenance Management, Kinnison and Sidiqi [20] recognize four main letter-checks currently being planned for aircraft: the A-check, the B-check, the C-check, and the D-check. The authors point out that many airlines have eliminated the B-checks by distributing their tasks to A- and C-checks. In the literature, there is a great variety of terminology for the different types of maintenance. Some scholars state that A-checks are short term checks that are to be executed biweekly [3], or even weekly [14, 40], whereas others state that they are to be executed roughly bimonthly. In the industry, the distribution of tasks over letter-checks and thus the size and interval of the checks vary between airlines as well. Some guideline data on the letter-checks from Kinnison and Sidiqi [20] is given in table 2.1. After discussions with employees and managers in leading European MRO providing companies, it is judged that this data is indeed on par with their definitions. Apart from the four main checks given in table 2.1, there also exist transit checks and daily checks. A transit check is performed before each flight and thus requires no planning effort. The daily check, despite its name, is not performed daily, but one to two times per week, depending on the aircraft type and the airline's definition of this check. A daily check includes a walk-around inspection, servicing of the oil, and a checkup on the lights and emergency equipment [27]. It is to allow for the daily check that the AMR is constructed to visit a maintenance station once every *d* days. For an A-check, an aircraft is usually taken out of operations for one day. A C-check will typically take between 4 and 7 days to complete [20].

Table 2.1: Standard check intervals and required man-hours, adapted from Kinnison and Sidiqi[20]

Check Interval and Man-Hours		
	Check Interval	Check Total Man-Hours
A-check	Every 600 FH	100
B-check	In 2 parts B1,B2 every 1200 FH	1200
C-check	In 2 parts C1, C2 every 5000 FH or 18 months	3420
D-check	First check done between 25k & 27.5k FH subsequent every 25k FH or 6 years	60,000

2.2. Scheduling Processes in the Airline Industry

A key function that influences the efficiency in operations is the scheduling of an airline's resources (i.e., aircraft and crew). The scheduling of an airline's operations and resources is an extremely complex and interrelated problem. Because of this complexity, it is the current practice to break up the resource planning into several stages. The stages in which the airline industry's different scheduling problems are divided are flight scheduling, fleet assignment, aircraft maintenance routing, crew scheduling, and the tail assignment [24].

Traditionally, these problems are solved sequentially, meaning that the result of the upstream problem is fed to the downstream problem [27]. Even though this approach does not lead to a globally optimal solution, it allows airlines to generate feasible solutions to their incredibly complex operational problems. An overview of an airline's sequential scheduling operations is given in figure 2.1.

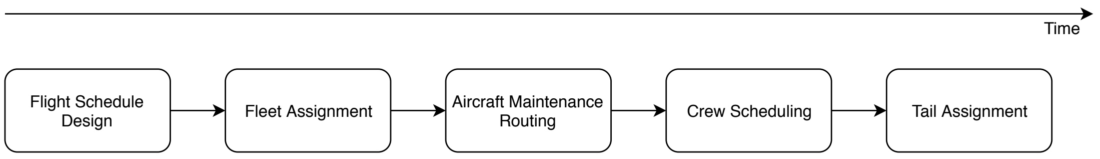


Figure 2.1: A schematic overview of an airline's scheduling operations

The flight schedule is perhaps the airline's most important operational planning task. The flight schedule is the primary product that an airline sells to customers and consists of a list of flight numbers that correspond to the origins, destinations, departure times, and arrival times [16]. The schedule is most often created approximately one year in advance by the marketing department, which considers several factors such as traffic forecasts, operations of competing carriers, internal resources and initiatives such as entering a new market [21].

The next stage in the overall scheduling framework is the fleet assignment (FA). During the FA, the flight schedule is taken as input, and the aim is to find the optimal assignment of equipment types for each of the flights. Typically, an airline's fleet consists of a variety of equipment types, such as the Airbus A320, Boeing 747, or Bombardier CRJ100. These equipment types differ considerably in seating capacity and various operational aspects. The objective of the fleet assignment is to maximize profit by preventing low load factors, such as when an aircraft with too large a capacity is selected, or potential spill of passengers to competitors exists, if the equipment type's capacity is lower than optimal [17].

Once each flight has been assigned an equipment type, the next step is to solve the aircraft maintenance routing (AMR). Given a fleet of a particular aircraft type and a set of flights, the goal of the AMR is to determine the flight routes for every aircraft such that the maintenance requirements, set by national and international authorities, are satisfied [27]. Only certain airports, named maintenance stations, are capable of performing maintenance operations for an airline [27]. It is easy to understand that for a hub-and-spoke network, the hub airport is usually the maintenance station. A feasible solution to the AMR is one where each constructed route visits a maintenance station at least once every *d* days so that a daily check can be performed. At this stage in the planning process, the routes are built for anonymous aircraft. The decision which tail number will fly which constructed route, is made closer to the day of operations during the tail assignment [22]. A tail number is a specific aircraft that is identified by its unique registration-number, often displayed on the tail of an aircraft, see figure 2.2. Note that the fleet assignment has decomposed the flight schedule, meaning there is an aircraft maintenance routing problem for every fleet of equipment types [21]. An example of an output provided by the AMR is given in figure 2.3.

After the aircraft routes are obtained, the crew scheduling (CS) is performed. Crew scheduling aims to assign crew members to the routes created in the AMR in order to minimize the crew cost and maximize various other objectives. These objectives include quality of life for the crew members and crew satis-

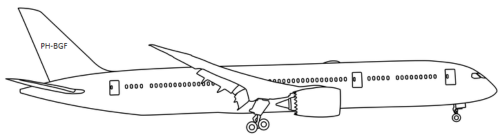


Figure 2.2: Side-view of an aircraft with its tail number: PH-BGF

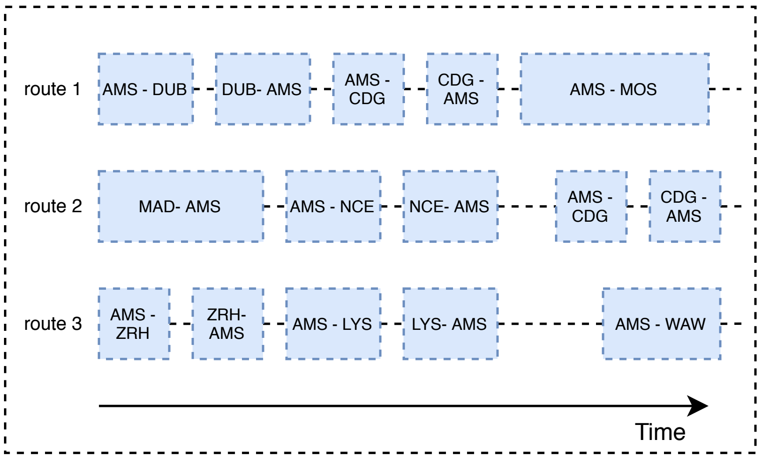


Figure 2.3: An example of the output of the AMR: a collection of flights that have been combined into routes. Note that only a small fraction of the created route is displayed in this figure due to space limitations.

faction [21]. The crew scheduling is usually conducted around one month before the day of operations, so that the crew has ample time to plan their lives outside of their working hours [13, 21]. An example of an output provided by the crew scheduling is given in figure 2.4.

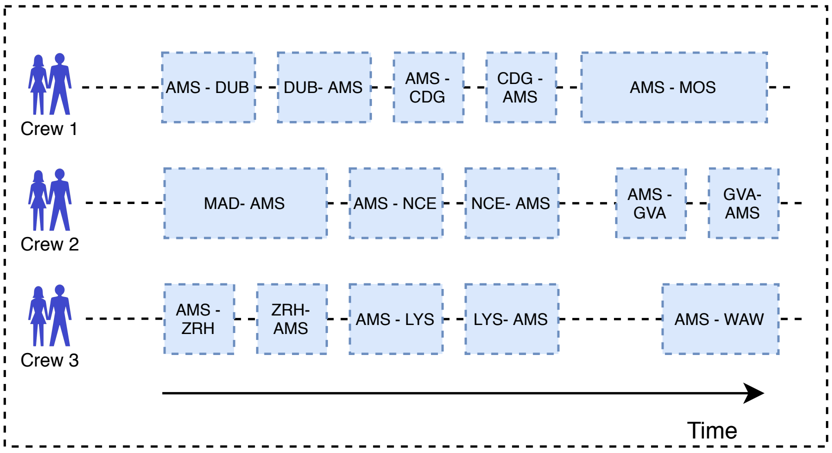


Figure 2.4: An example of the output of the CS: crew members are assigned to routes created in the AMR

Only a few weeks or even days (this varies strongly between airlines) before the day of operations, tail numbers are assigned to the routes, constructed in the AMR [21]. During this assignment, the planners must ensure that each route is assigned to exactly one tail number, that the tail number is capable of carrying out the assigned route and that the route is feasible with respect to the tail number's individual maintenance requirements. To achieve this, the aircraft's location and the maintenance and flying history at the start of the horizon are considered [35]. The planners might need to make adjustments to the routes because these considerations and possible schedule disruptions could result in an infeasible maintenance schedule. When a tail number has been assigned a route, all maintenance activities must be scheduled during the available ground times it will spend on maintenance stations. An example of an output provided by the tail assignment is given in figure 2.5. The green arrows indicate a proposed swap, which might be beneficial if aircraft 2 would benefit from a longer ground time on Amsterdam for maintenance. In this example, Amsterdam is the maintenance station.

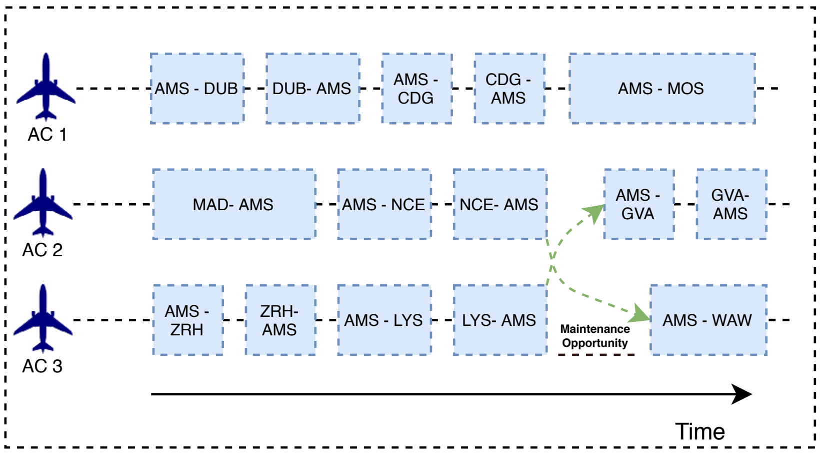


Figure 2.5: An example of the output of the TA: tail numbers are assigned to the routes created in the AMR. An aircraft swap is proposed between aircraft 2 and aircraft 3.

Many airlines operate from a hub-and-spoke system, where the hub often is the only maintenance station. Therefore, to optimize the maintenance schedules for aircraft, it is essential to assign routes to them that visit the maintenance station at the correct time. This is done in the TA by adjusting the routes that were created in the AMR, as was shown in figure 2.5. However, when adjusting the routes, the feasibility of the crew rosters must not be compromised. If the swap in figure 2.5 were to be executed, crew 2 and crew 3 would have to change aircraft during the ground time at Amsterdam. This crew-swap increases the probability of extra delays because the delay of one aircraft would propagate throughout the network. In fact, to avoid spreading delays in the network, crews should stay with the same aircraft as much as possible [10], or crews should only swap during extensively long ground times. The most common of these extensively long ground times is an overnight ground time. Therefore, swapping aircraft during an overnight ground time poses little risk of a propagated delay [27].

2.2.1. Aircraft Maintenance Scheduling in the Industry

It is concluded from interviews with managers and engineers in leading European MRO-providing companies, that most airlines still aggregate high-interval maintenance tasks within letter-checks. Smaller and low-interval tasks are performed within daily checks, usually during an overnight stay at a maintenance station. Letter-checks are planned within specific pre-made slots. For example, throughout the year, there are multiple A-check slots, spanning an entire day, which can be claimed for specific aircraft. These slots are claimed roughly one month in advance. If it is expected that a tail number will have an A-check due in a month, the planners try to make sure that the tail number will claim one of these slots. The TA's planning horizon varies between airlines but is usually substantially less than a month. This means that a tail number does not yet know the complete route it will travel, only that it is expected at the maintenance station on the day of the A-check. As time goes on and the planned A-check moves inside the planning horizon, the aircraft is given a route, which makes sure that it will arrive at the maintenance station at the right time. A schematic overview with a realistic time frame is given in figure 2.6. C- and D-checks are planned even further in advance, as they have much higher intervals than the A-check and require more down-time. As stated in chapter 1, the main advantage of using letter-checks lies in the relative ease of the planning. The main disadvantages are the need to take the aircraft out of rotation and the fact that many maintenance tasks are performed before they are due.

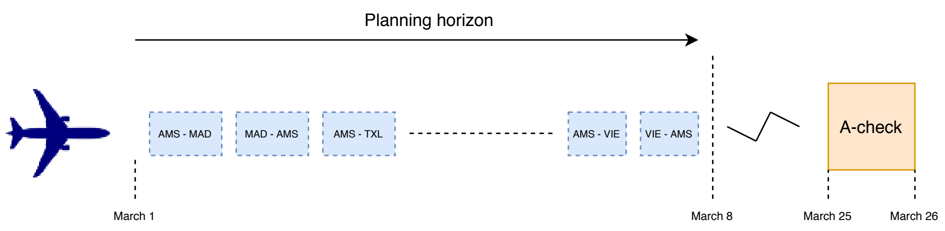


Figure 2.6: An example of the time frame in which an A-check is currently scheduled in the airline industry.

2.2.2. Aircraft maintenance Scheduling in the Literature

Maintenance scheduling in the airline industry is intertwined with routing because airlines typically have a limited number of maintenance stations. Therefore, assuring that the aircraft is in the right position at the right time is fundamental to the problem. This is why the Aircraft Maintenance Routing (AMR) and the Tail Assignment (TA) are the main focuses of studies that have focused on the optimization of aircraft maintenance scheduling. In this thesis, the AMR is defined as the construction of non-tail assigned flight routes from individual flights. The TA is defined as the process during which these routes are taken as input and are assigned to tail numbers. The adaption of the routes that are given as input is also part of the TA since this might be necessary to satisfy operational constraints. Some works in the literature have attempted to build tail number-specific routes from individual flights. In this thesis, that work, which is sometimes named the operational aircraft maintenance routing (OAMR), is regarded as an integration between the AMR and the TA.

Table 2.2 presents an analysis of research published on maintenance scheduling and routing problems. The covered areas are the fleet assignment, the aircraft maintenance routing, and the tail assignment. A distinction is made between a cyclical maintenance/routing schedule (CS) and one that operates with a finite horizon (FH). Furthermore, distinctions between papers are made in terms of whether their maintenance requirements are check-based (CB), where maintenance is handled as a generic check, or task-based (TB), where tasks are considered at a more individual level. Finally, papers are assessed on whether or not they are compliant with an airline's overall scheduling framework (COF), as was discussed in section 2.2. A selection of the presented papers is discussed below.

Table 2.2: Analysis of articles published on related maintenance scheduling and routing problems

Reference	Covering Areas			Schedule		Maint. Considerations		
	FA	AMR	TA	CS	FH	CB	TB	COF
Feo & Bard [14]		√		√		√		√
Barnhart et al. [7]	√	√		√		√		
Gopalan & Talluri [16]		√		√		√		√
Moudani & Mora-Camino [32]		√	√		√		√	
Sriram & Haghani [40]			√	√		√		√
Sarac et al. [37]		√	√		√		√	
Liang et al. [26]		√		√		√		
Liang & Chaovalitwongse [25]	√	√		√		√		
Basdere & Bilge [8]		√	√		√	√		
Maher et al. [30]		√	√		√		√	
Liang et al. [27]		√	√	√		√		
Ruther et al. [35]		√	√		√		√	
Khaled et al. [19]		√	√		√	√		
Safaei & Jardine [36]		√	√		√		√	
This thesis			√		√		√	√

FA: Fleet Assignment, AMR: Aircraft Maintenance Routing, TA: Tail Assignment, CS: Cyclic Schedule, FH: Finite Horizon, CB: Check-Based, TB: Task-Based, COF: Compliant with Overall Framework.

Papers that build a cyclical schedule using day-routes. Feo & Bard [14] and Gopalan & Talluri [16] have focused on solving the AMR by creating weekly cyclical flight schedules that spend the night at a maintenance station at least once every four days. They both use a closed-loop city-day-network in which cities are represented by nodes, and the arcs between the nodes represent day-routes. A day-route contains all the flights an aircraft makes that day and is connected to the nodes that represent the cities from which the first flight of the day departs and at which the last flight of the day arrives. Their objectives are to find maintenance feasible cyclical schedules by connecting day-routes with each other and by making sure that the last flight of the last day-route arrives at the same station as the first flight from the first day-route. The advantage of a cyclical schedule is that, in theory, it could be executed in perpetuity. An additional objective of Feo & Bard was to decrease the number of required maintenance stations. Sriram & Haghani [40] take day-routes as input and solve the TA by creating tail-specific weekly cyclical flight routes. They use a matrix, representing the cost of maintenance per aircraft per city, to try and find a minimal cost route while adhering to the constraint of spending the night on a maintenance station at least once every four days. Since all three works use day-routes, they are all compatible with an airline's overall planning framework, as discussed in section 2.2.

Papers that solve a check-based OAMR. Basdere & Bilge [8] solve the OAMR over a weekly planning horizon with the goal of maximizing the utilization of the total remaining flying time by planning the generic maintenance check as late as possible. Their approach considers all ground times as possibilities for maintenance. A drawback to their model is that aircraft can undergo maintenance at most once during the planning horizon. Khaled et al. [19] solve the OAMR in a compact model, which allows them to operate a planning horizon longer than one week.

Papers that solve a task-based OAMR. Sarac et al. [37], Maher et al. [30], Moudani & Mora-Camino [32], Ruther et al. [35], and Safaei & Jardine [36] not only consider maintenance requirements as a generic check that should be executed once every *d* days but propose models that can handle different kinds of maintenance tasks. Sarac et al. [37] and Maher et al. [30] model the daily OAMR. In Sarac et al. [37], if an aircraft is labeled high-time, which means that it might require maintenance, of any kind, at the end of the day, it is routed in such a way that it will spend the same night on a maintenance station. Moudani & Mora-Camino [32] solve an OAMR for a charter airline in which maintenance tasks are planned after the creation of the routes, in the resulting ground times. If no feasible solution can be found, it must be anticipated which flights should be delayed in order to create a feasible gap for the maintenance. The OAMR model proposed by Safaei & Jardine [36] attempts to minimize the mismatch between tail numbers' maintenance needs and maintenance opportunities using an iterative heuristic. Ruther et al. [35] solve an integrated OAMR and crew pairing four days before the day of operations. Following such operations, the crew would be informed ample time in advance when they would be working, but only four days in advance which flights. As they state themselves, this approach might be challenging to implement for larger airlines but could be interesting to low-cost carriers.

Many studies in the literature focus on finding a cyclical flight schedule, in which the departure station on the first day and the arrival station on the last day of the schedule's horizon are the same. If sufficient maintenance operations are planned within the cyclic schedule, it could be executed in perpetuity. In practice, using a single rotation is not applicable due to the stochastic nature of operations in the airline industry [8]. This stochastic nature is why operational models use a finite planning horizon to construct and assign the routes.

There are two ways in which the maintenance requirements are modeled in the literature. The first way uses a check-based strategy, meaning that a generic maintenance check should be scheduled at least every *d* days or *f* flight hours. The second way utilizes a task-based strategy, where multiple maintenance tasks, with their specific due dates, for individual aircraft, are considered.

Most research that has focused on solving the OAMR planned their routes for a one-week time horizon or shorter, while in practice, the crew rosters are usually published roughly a month in advance. This means that in order to be able to stick with their rosters, crew members would most likely have to swap from aircraft after a flight numerous times. Necessitated crew-swapping increases the probability of extra delays since the delay of one aircraft would now propagate throughout the network. Therefore, it is concluded that they are not compliant with the overall planning framework of airlines.

2.3. Mathematical Formulations for Aircraft Maintenance Scheduling and Routing

The most commonly used models to formulate the maintenance scheduling and routing problems in the literature are the network-based models and the string-based models, of which the network-based models can be divided into City-Day Networks (CDN), Time-Space Networks (TSN) and Connection Networks (CN) [36]. Their definitions, their main limitations, and relevant papers using these models are presented in table 2.3.

Table 2.3: Discussion of the most commonly used model formulations of the aircraft maintenance scheduling and routing problems in the literature

Model	**Definition**	**Main limitation**	**Relevant papers**
City-Day Networks (CDN)	Network structure in which nodes represent cities and arcs represent the flights or routes between them.	Difficult to keep track of arrival and departure times of arcs within the network [37].	[40] [16] [14]
Time-Space Networks (TSN)	Network structure in which each station is represented by a timeline, which consist of a sequentially ordered series of event nodes to represent flight arrivals or departures [25].	The connections between flights are not explicit. This makes it impossible to trace the routes of individual aircraft [7].	[24] [26] [25]
Connection Networks (CN)	Network structure in which nodes represent flight legs and arcs represent feasible connections among the flight legs [37].	Number of arcs progressively increases with size of the network [36].	[37] [8] [30] [35] [19] [36]
String-based models	Model structure in which a string represents a sequence of connected flights that begins and ends at a maintenance station which satisfies the flow balance and is maintenance feasible [7].	The number of strings grows exponentially with the size of the flight network and the fleet [36].	[7] [27]

2.4. Discussion of the Literature Review

An airline solves its scheduling operations in a sequential manner. First, it creates the flight schedule, and after this, aircraft types are assigned to the flights. For each of the aircraft types, non-tail number specific routes are generated during the AMR, to which the crews are assigned first, and in a later stage, the tail numbers as well. During this tail assignment, the routes may be altered, and the maintenance is planned within them. Nowadays, the industry still largely depends on the use of extensive letter-checks during which aircraft are taken out of operations, and many maintenance tasks are executed together.

The maintenance planned within the flight schedules consists mostly of smaller and lower-interval tasks and is congregated in daily checks. Based on the conducted literature review and the papers presented in table 2.2, it can be seen that much of the literature on aircraft maintenance planning and routing is limited to planning generic daily checks. The papers that do consider a more task-based oriented approach are not compliant with the airline's overall planning framework, leaving them vulnerable to the risk of propagating delays resulting from the necessary crew swaps. It is therefore concluded that there is a gap on the subject of the optimization of the operational routing and maintenance scheduling of aircraft on an individual task basis in a way that complies with an airline's overall planning framework. In addition, apart from a case study by Senturk and Ozkol [38, 39], no research was found on the extent of the advantages that an individual task-based maintenance planning approach holds.

The use of day-routes, as used by Feo & Bard [14], Gopalan & Talluri [16] and Sriram & Haghani [40], is a well working method by which the solutions of the AMR could be altered in the TA without exposing the airline to the risk of propagated delays because an overnight stay at a station gives ample time for a crew swap to take place. City-Day networks are the best choice when using day-routes because their main limitation, the difficulty it has keeping track of arrival and departure times of arcs within the network, only becomes a concern when one has to build routes from individual flights. Time-Space networks are not fit for our application as it is essential in the TA phase to be able to trace the routes of individual aircraft. Connection networks and string-based networks could be applicable, but their model formulations contain many more arcs, leading to a much larger model and a much higher number of decision variables.

3 Problem Definition

The maintenance scheduling and routing model in this thesis can be defined as follows. Given a flight schedule containing *F* flights, the objective of the model is to find a route for each tail number such that (a) all the flights are covered; (b) each tail number spends a night at a maintenance station at least once every *d* days to allow for necessary daily checks; (c) high-interval maintenance tasks are individually planned; (d) initial conditions for each tail number regarding location and maintenance needs are taken into account; (e) the process fits inside the airline's overall planning framework; (f) the cost of maintenance is minimized and (g) the capacity of the maintenance station is taken into account.

In section 2.2.2, we indicated that the use of day-routes is an effective method to solve the tail assignment in a way that complies with the overall scheduling framework of an airline. By partitioning day-routes to tail numbers, the only required aircraft swaps that a crew has to make are during an overnight stay, which gives very little chance of propagating delays [27]. Gopalan and Talluri [16] refer to day-routes as *Lines of Flying* (LOFs), which is a term that will be adopted in this thesis. LOFs specify the origin at the start of the day and the destination at the end of the day of a route. Thus, for example, if an aircraft departs at the start of the day from Amsterdam, flies to Berlin, flies back to Amsterdam and then to Madrid as its final flight of the day, the information is summarized as an LOF from Amsterdam to Madrid. An example of a city-day network with a planning horizon of seven days is visualized in figure 3.1, where the flight schedules of two aircraft are given. The arcs in figure 3.1 represent the LOFs.

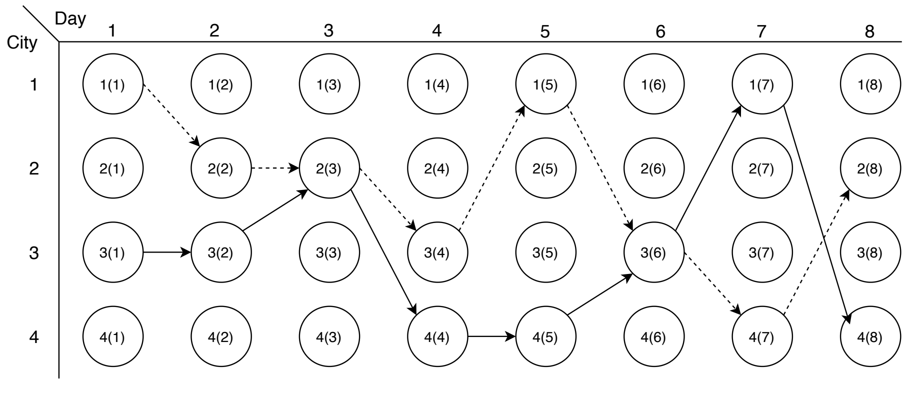
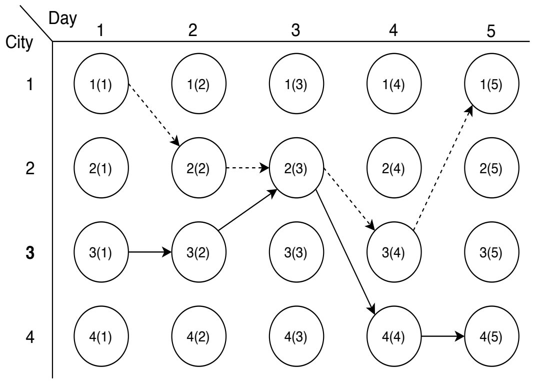
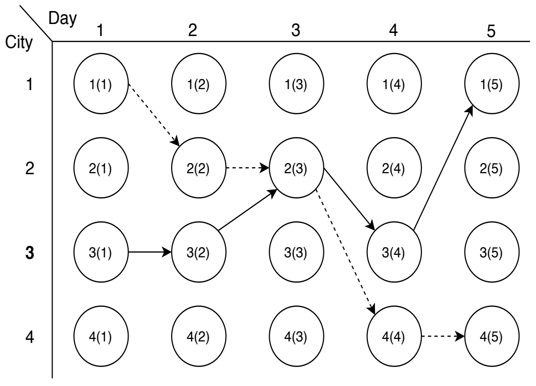


Figure 3.1: City-day network with a 7 day planning horizon

During the AMR, the non-tail number specific routes are constructed to ensure that each route spends at least one night per given interval at a maintenance station to allow for necessary daily checks. If these generic routes are broken down into LOFs, this warrant no longer holds because aircraft can swap day-routes when spending the night at the same station. An example of how this could perturb the maintenance feasibility is shown in figures 3.2a and 3.2b. Figure 3.2a shows a city-day network, including two routes created in the AMR. In this situation, city 3 represents the maintenance station, and a daily check interval of four days is prescribed. The two routes, created in the AMR, honor the constraint of spending a night at the maintenance station at least once per given interval. The created routes in figure 3.2a are broken up into LOFs, and the LOFs are assigned to the tail numbers piece-wise. As both aircraft depart from 2(3), the given formulation allows them to swap, creating a possibly infeasible route for one of the aircraft, shown by the dashed route in figure 3.2b.



(a) Routes as they were created in the AMR



(b) Routes after the swapping of LOFs on 2(3)

Figure 3.2: The possible perturbation of maintenance feasibility using day-routes

This infeasibility problem could be overcome by adding another routing constraint that forces each aircraft to spend at least one night per interval at a maintenance station. However, adding this as a separate constraint severely increases the size of the model. Therefore, we choose to combine the LOFs into new lines-of-flying so that each line starts and/or ends at a maintenance station. To keep things unequivocal, we will refer to these new creations as *lines*, so that the term LOF can keep its previously established definition of a one-day route.

Because our model is an operational one and works with a finite horizon, it is unlikely that each line can start and end at a maintenance station. At the start or end of the time horizon, an aircraft may be positioned at a non-maintenance station. In these cases, a line will either only begin or end at a maintenance station. An example of how the lines are formed is demonstrated in figure 3.3, in which city 3 again represents the maintenance station. The LOFs that were present in figure 3.1 are now combined into lines that start and/or end at the maintenance station. Each color in figure 3.3 represents a separate line. Regardless of the length of the time horizon or what city a line starts or ends at, the solution produced in the AMR guarantees that a line will never go more than a set number of days without spending the night at a maintenance station.

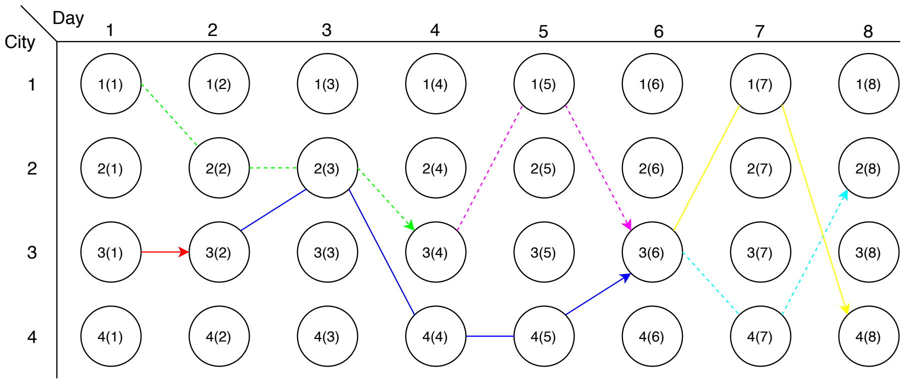


Figure 3.3: The formed lines within the illustrated city-day network

When using lines instead of LOFs, the only requirement for guaranteeing maintenance feasibility is that the first assigned line for each aircraft visits a maintenance station before the due date of its daily check. An example of this is given in figure 3.4. In this situation, two separate aircraft start from 2(1). If one of the aircraft is due for its necessary daily check on the maintenance station (station 3) by day 2, it must be assigned the red line and not the green line. A constraint guaranteeing that the first selected line leads to a feasible solution must be added to the model. The existence of the AMR ensures that a feasible line, such as the red one, exists for each aircraft.

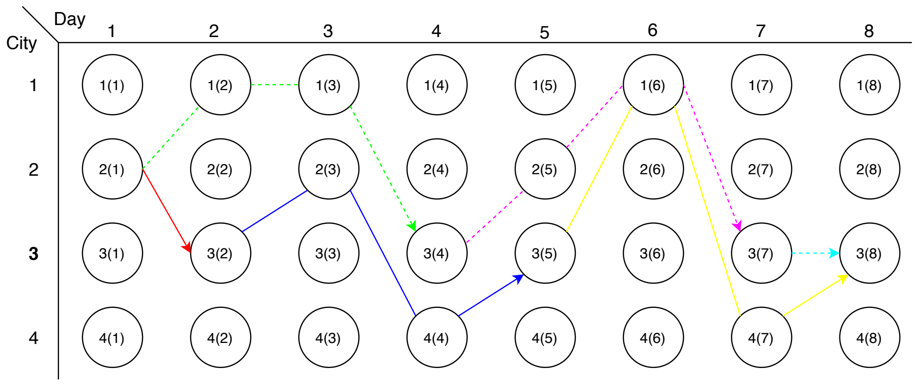


Figure 3.4: City-day network in which multiple aircraft start from the same node

Apart from the routine daily checks, also tasks with a high interval that would generally be planned inside of a letter-check are considered in our model. It is unusual for an airline to have all aircraft of a particular fleet undergo a letter-check in the same week. After all, this would result in severe harm to the airline's operability as many aircraft would have to be taken out of operations during the same time. An airline would instead attempt to spread this out over the length of the interval of the letter-check. By moving high-interval tasks from a letter-check to an overnight check, these tasks would be executed during the aircraft's flight schedule, thereby reducing the need for aircraft to be taken out of operations and increasing aircraft availability and revenue. As a result of the phasing of the previous letter-check strategy, we expect that some aircraft will have a higher number of tasks that need to be scheduled during the planning horizon, and some aircraft will require a lower number of tasks to be scheduled. By assigning a route consisting of multiple short lines and thus multiple maintenance visits instead of only assigning the minimum, an aircraft with high maintenance demand can be given the capacity to handle all its maintenance requirements. The maintenance routing and scheduling model presented in this thesis will use the partitioning of lines to tail numbers to identify such a schedule.

The tasks will be provided with both a due date and a remaining numbers of legal flight hours that can be flown before execution of the task is mandatory. If either the due date or the remaining number of legal flight hours is exceeded before the task is executed, the aircraft loses its airworthiness. This means that whichever of these two limits appears first determines the cut-off point before which a task must be executed. Since it is not known beforehand how many flight hours each tail number will fly, it is also unknown which specific set of tasks will need to be planned within the time horizon. To account for this, the longest possible route for each tail number is determined. All the maintenance tasks that, at the start of the time horizon, have a remaining number of legal flight hours smaller than the number of flight hours in the longest route are included in the model. This does not mean that all tasks must be executed, because the actual traveled route may be shorter. A similar idea was also implemented by Bilge & Basdere [8], who labeled all aircraft with a check due in less remaining time than the longest possible route in the planning horizon as high-time aircraft, which then were considered for maintenance, but were not forced to undergo it if it turned out unnecessary.

In theory, an aircraft's route could be infeasible if one of the individual tasks is due before its first chance to visit a maintenance station appears. However, the tail assignment problem is generally solved with a rolling horizon [23], which means that this problem would be solved beforehand by planning and performing the maintenance task at an earlier stage.

The clear objective of most maintenance scheduling models is to minimize the cost of maintenance. However, because it is difficult to define and calculate a cost for maintenance, often a surrogate objective is used [37]. If a maintenance task with positive remaining flying time is executed, then a portion of the flight capacity is wasted, and if necessary, parts that need to be replaced are done so before utilizing their useful lives [8]. To minimize this wasted flying time is equivalent to minimizing the unused legal flying times. Minimizing the unused flying time is taken as a surrogate objective for the cost minimization in this work, similar to Sarac et al. [37] and Basdere & Bilger [8].

The problem is formulated as a multi-commodity network flow model, where each aircraft represents a separate commodity. The created lines in the AMR are taken as an input, and each line has an upper and lower capacity of one unit flow. The proposed model is concerned with assigning the lines to tail numbers in a way that allocates enough maintenance opportunities to tail numbers that require it and tries to plan the individual tasks as late as possible.

Our model considers a hub-and-spoke system in which the hub is the only maintenance station. Also, similarly to Feo & Bard [14], Gopalan & Talluri [16], and Sriram & Haghani [40], an intracontinental fleet is considered, because intercontinental aircraft do not operate with lines as they also fly during the night [14]. In a hub and spoke system, aircraft operate flights only to and from the hub.

The assumptions made in the proposed formulation are summarized as:

1. The planned aircraft maintenance is only performed during the night
2. There is no aircraft operation during the night
3. The model considers a hub-and-spoke model where there is only one maintenance station
4. The given station capacity is fully available for the execution of the planned tasks

Night flying restrictions are common in European airports, as many large airports have installed curfews. One of the main reasons for this is to relieve communities in close proximity to an airport from noise pollution [43]. Frankfurt airport has a curfew of 23:00 to 05:00 [33], London Heathrow has no scheduled departures between 22:50 and 06:00 [2] and Schiphol Airport has restricted take-offs and landings between 23:00 and 06:00 to a single take-off runway and a single landing runway [1]. Because intracontinental routes have a night curfew, most inspections and repairs (other than transit checks) as a general rule take place at night [19].

3.1. Mathematical Formulation

This section presents the mathematical formulation. The definitions behind the notation used in this model can be found in table 3.1.

$$Minimize \quad \sum_{i\in I}\sum_{j\in J}\sum_{t\in T_i} y_{itj} * S_{i,t} * min(Avg * (DD_{\text{i,t}} - Date_{\text{j}}), RFH_{\text{i,t}} - \sum_{d=1}^{Date_{\text{j}}}\sum_{j\in Js_{\text{d}}} x_{\text{ij}} * l_{\text{j}}) \tag{3.1}$$

Subject to:

$$\sum_{i\in I} x_{ij} = 1 \qquad \forall j \in J \tag{3.2}$$

$$\sum_{j\in Ja_d} x_{ij} = 1 \qquad \forall i \in I, d = 2, 3, ..., n_{\text{d}} \tag{3.3}$$

$$\sum_{j\in S_i} x_{ij} = 1 \qquad \forall i \in I \tag{3.4}$$

$$x_{ij} - \sum_{j \in C_j} x_{ij} \leq 0 \qquad \forall i \in I, j \in J_b \tag{3.5}$$

$$\sum_{j \in J} y_{itj} \leq 1 \qquad \forall i \in I, t \in T_i \tag{3.6}$$

$$RFH_{i,t} - \sum_{j \in J} x_{ij} * l_j \geq \sum_{j \in J} y_{itj} * ms_j * -K \qquad \forall i \in I, t \in T_i \tag{3.7}$$

$$DD_{i,t} - n_d \geq \sum_{j \in J} y_{itj} * ms_j * -K \qquad \forall i \in I, t \in T_i \tag{3.8}$$

$$y_{itj} * (\sum_{d=1}^{Date_j} \sum_{j \in Js_d} x_{ij} * l_j) \leq RFH_{i,t} \qquad \forall i \in I, t \in T_i, j \in J \tag{3.9}$$

$$y_{itj} * Date_j \leq DD_{i,t} \qquad \forall i \in I, t \in T_i, j \in J \tag{3.10}$$

$$|T_i| * x_{ij} \geq \sum_{t \in T_i} y_{itj} \qquad \forall i \in I, j \in J \tag{3.11}$$

$$\sum_{i \in I} \sum_{j \in Je_d} \sum_{t \in T_i} y_{itj} * S_{i,t} \leq M_d \qquad d = 1, ..., n_d \tag{3.12}$$

$$x_{ij} \in \{0, 1\} \qquad \forall i \in I, j \in J \tag{3.13}$$

$$y_{itj} \in \{0, 1\} \qquad \forall i \in I, t \in T_i, j \in J \tag{3.14}$$

3.1.1. Objective Function

The objective function is set up to minimize the unused flying time of the scheduled tasks. Herein the size of the task is taken as a weight factor. This stems from the idea that it is better to fully utilize the task interval of large tasks than smaller ones. The unused flying time for a task is calculated in two ways, both for unused days and for unused flying hours. The former determines the due date of the task and compares it to the date that the task was executed. This is then multiplied by the average number of flight hours that an aircraft records per day, which is given as an input parameter to the model. The latter way of determining the unused flying time compares the remaining number of legal flight hours at the start of the planning horizon with the number of flight hours the aircraft has recorded until the execution of the task. As both the due date and the remaining number of legal flight hours of a task set a hard line after which the aircraft loses its airworthiness, the minimum value of these two is selected as the truly lost flying time.

$$Minimize \quad \sum_{i \in I} \sum_{j \in J} \sum_{t \in T_i} y_{itj} * S_{i,t} * min(Avg * (DD_{i,t} - Date_j), RFH_{i,t} - \sum_{d=1}^{Date_j} \sum_{j \in Js_d} x_{ij} * l_j) \tag{3.15}$$

Table 3.1: Parameters, indexes, sets and decision variables

Parameters & Indexes:	
n_p	number of aircraft in the fleet
n_d	number of days in the planning horizon
n_j	number of lines in the planning horizon
$n_{i,t}$	number of considered tasks in the planning horizon for aircraft *i*
i	index for aircraft $i = 1,2,...,n_p$
d	index for days $d = 1,2,...,n_d$
j	index for lines $j = 1,2,..,n_j$
t	index for tasks *t* of aircraft *i*, $t = 1,2,...,n_{i,t}$
l_j	length of of line *j* in flight hours
$DD_{i,t}$	due date of task *t* of aircraft *i*
$RFH_{i,t}$	remaining number of legal flight hours of task *t* of aircraft *i* at the start of the planning horizon
$Date_j$	end date of line j
$S_{i,t}$	size in man-hours of task *t* of aircraft *i*
ms_j	equals 1 if line *j* ends on the maintenance station, 0 otherwise
M_d	Capacity of the maintenance station in man-hours on day d
Avg	average number of recorded flight hours per day per aircraft
K	a sufficiently large number
Sets:	
I	set of all aircraft
J	set of all lines in the planning horizon
Ja_d	set of all lines active on day *d*
Js_d	set of all lines starting on day *d*
Je_d	set of all lines ending on day *d*
J_b	set of all lines except the ending lines
S_i	set of all feasible starting lines for aircraft *i*
C_j	set of all connection lines of line *j*
T_i	set of considered tasks for aircraft *i* within the planning horizon
Decision Variables:	
x_{ij}	= 1 if aircraft *i* flies line *j*, 0 otherwise
y_{itj}	= 1 if aircraft *i* plans maintenance task $t \in T_i$ after line *j*, 0 otherwise

3.1.2. Set of Constraints

Constraints 3.16 - 3.19 are concerned with *creating a feasible route for each tail number*. Constraints (3.16) are known as the *flight coverage* constraints. They guarantee that each line, and thus each individual flight, is allocated to exactly one aircraft.

$$\sum_{i \in I} x_{ij} = 1 \qquad \forall j \in J \tag{3.16}$$

Constraints (3.17) limit the aircraft to fly at most 1 line at any given moment, as it is clear that an aircraft cannot be in more than one place at the same time. Furthermore, these constraints are only set for day 2 until the last day of the time horizon, since day 1 requires a separate set of constraints.

$$\sum_{j \in Ja_d} x_{ij} = 1 \qquad \forall i \in I, d = 2, 3, ..., n_\mathrm{d} \tag{3.17}$$

The first assigned line in every route is one that requires extra consideration. Firstly, a tail number must be assigned a route that starts from its initial location. An aircraft located in Amsterdam, should not be given a flight schedule departing from Madrid for obvious reasons. Secondly, the due date for the first mandatory overnight stay to complete a daily check, at the maintenance station, must be taken into account. The first selected line in the route should end at a maintenance station before this due date passes. Therefore, for each tail number, a set S_i is made that contains all the feasible starting lines for aircraft *i*. Constraints (3.18) ensure that each aircraft is allocated exactly one line that starts from its corresponding set S$_i$.

$$\sum_{j \in S_i} x_{ij} = 1 \qquad \forall i \in I \tag{3.18}$$

Constraints (3.19) are the last in the routing section, and are the *flow* constraints. These constraints ensure that if an aircraft flies a line, it also must fly a line connected to it. For each line, all its connections are stored in set C$_j$. Therefore, if line 10 arrives at Amsterdam on day 2, C$_{10}$ contains all the lines that will depart from Amsterdam on day 3. The lines ending on the last day are excluded since they are the endpoint in the planning horizon and thus fall outside the planning horizon.

$$x_{ij} - \sum_{j \in C_j} x_{ij} \leq 0 \qquad \forall i \in I, j \in J_b \tag{3.19}$$

Constraints 3.20-3.26 deal with the *planning of the individual high-interval maintenance tasks for each aircraft*. As stated at the start of this chapter, these tasks are given both due dates and remaining numbers of legal flight hours. Before the model is solved, it is not yet known how many flight hours will be flown by each tail number. This means that it cannot be said with certainty which set of tasks needs to be planned for each tail number in the planning horizon. Therefore, the longest possible route for each tail number is determined and all the tasks that would have to be planned for this tail number, if this route were to be the flown, are considered in the model and saved in set *T$_i$*. If a shorter route is constructed, some tasks may fall outside the planning horizon, and therefore constraints (3.20) do not require all tasks to be planned exactly once, but instead, they ensure each task can be planned a maximum of one time.

$$\sum_{j \in J} y_{itj} \leq 1 \qquad \forall i \in I, t \in T_i \tag{3.20}$$

Constraints (3.21) ensure that if the constructed route for aircraft *i* is longer than a task's remaining number of legal flight hours, it must be planned on a maintenance station.

$$RFH_{i,t} - \sum_{j \in J} x_{ij} * l_j \geq \sum_{j \in J} y_{itj} * ms_j * -K \qquad \forall i \in I, t \in T_i \tag{3.21}$$

Similar to constraints (3.21), constraints (3.22) ensure that a task must be planned if its due date is nearer than the end of the planning horizon.

$$DD_{i,t} - n_d \geq \sum_{j \in J} y_{itj} * ms_j * -K \qquad \forall i \in I, t \in T_i \tag{3.22}$$

Constraints (3.23) ensure that, *if* a maintenance task is planned, it must be planned before the remaining number of legal flight hours for that task have expired.

$$y_{itj} * (\sum_{d=1}^{Date_j} \sum_{j \in J_{sd}} x_{ij} * l_j) \leq RFH_{i,t} \qquad \forall i \in I, t \in T_i, j \in J \tag{3.23}$$

Similar to constraints (3.23), constraints (3.24) ensure that, *if* a maintenance task is planned, it must be planned before its due date.

$$y_{itj} * Date_j \leq DD_{i,t} \qquad \forall i \in I, t \in T_i, j \in J \tag{3.24}$$

Constraints (3.25) assure that a task can only be planned for an aircraft after a line *if* the aircraft in question has actually flown that specific line.

$$|T_i| * x_{ij} \geq \sum_{t \in T_i} y_{itj} \qquad \forall i \in I, j \in J \tag{3.25}$$

Constraints (3.26) are the *capacity* constraints. All aircraft in the model have a shared resource in the maintenance station's capacity. The maintenance station is allocated a number of man-hours it has at its disposal for handling the tasks taken as input to the model. The sum of all executed maintenance tasks over all aircraft per night must not exceed this number.

$$\sum_{i \in I} \sum_{j \in J_{ed}} \sum_{t \in T_i} y_{itj} * S_{i,t} \leq M_d \qquad d = 1, ..., n_d \tag{3.26}$$

3.2. Model Characteristics

The presented model is a large scale MIP. To gain an appreciation for its size, a breakdown of the problem sizes and computational results for different sets of parameter settings is presented in table 3.2. The given tests are run with a planning horizon of 7 days, which is a common planning horizon in the literature [31]. Longer planning horizons could yield higher quality routes from an optimization perspective, but the stochastic nature of the airline environment makes it less likely that these routes could be fully carried out [8]. Selecting a planning horizon that is too short can lead to feasibility problems due to large task sets suddenly entering the planning horizon. Therefore, in our tests, a planning horizon of one week, or seven days, is selected similarly to Basdere & Bilge [8]. Furthermore a maintenance capacity of 75 man-hours per night at the maintenance station is selected, and in our test the maintenance task sets for each aircraft consisted of either 10 or 20 tasks.These task set sizes were chosen as they are in the range of the number of tasks that could be expected to require planning every week if the tasks from letter checks are planned individually. The model was coded in Python 3.7 and solved using Gurobi, who employ a branch-and-bound based algorithm. All computations have been performed on a desktop running Intel(R) Core(TM) i7-7500U CPU @ 2.70 GHz.

Table 3.2: Computational results when solving the MIP, with n_d =7, using Gurobi

# AC	# Tasks	# Var	# Const	Residual Gap (%)	CPU (sec)	Obj value
5	10	1045	2276	0.00	1	258
	20	1995	4326	0.00	2	471
6	10	1452	3119	0.00	1	304
	20	2772	5939	0.00	7	607
7	10	1925	4092	0.00	5	385
	20	3675	7802	0.00	14	727
8	10	2464	5195	0.00	19	424
	20	4704	9915	0.00	44	863
9	10	3168	6627	0.00	53	434
	20	6048	12,657	0.00	110	913
10	10	4070	8454	0.00	632	432
	20	7770	16,154	0.00	2138	937
11	10	4840	10,013	0.00	1269	471
	20	9240	19,143	5.41	56,000	1085

The flight data for these tests was extracted from online sources tracking individual aircraft of fleets. In this case, the flight data of a set of Boeing 737s of a large European airline was used. For each tail number, the recorded flight data over a specified time horizon was taken, and these flights were transformed into the lines we use in the model. The starting positions of the tail numbers are also extracted from this data. If a test case of five aircraft was used, only the flight data of these five aircraft over the given time horizon would be used. Adding another aircraft to the model means that the total number of lines also expands. For certain aircraft, the flight data was complemented if a gap in the data were present.

All aircraft starting from the maintenance station have been given a due date for the first daily check of four days from the start of the planning horizon. For aircraft not starting from the maintenance station, the first due overnight is generated following a discrete uniform distribution. Within this distribution, the lower bound is set as the first time the aircraft spends the night at the maintenance station in the flight data. This guarantees the existence of a feasible solution. The upper bound is given as three days from the start of the planning horizon.

Each aircraft has been given a unique set of tasks to be completed. The size of each task in man-hours is generated following a uniform distribution between 0.5 man-hours and 1.5 man-hours so that the average lies around 1 man-hour, which in section 2.1.2 is concluded to be the average size of a task in an A-check. Each task's due date is generated following a discrete uniform distribution. The lower bound of this distribution is the due date for the aircraft's first daily check, and the upper bound is the end of the planning horizon. The number of remaining legal flight hours for a task is generated following a discrete uniform distribution where the lower bound equals the number of flight-hours that are flown before the aircraft can its first daily check. This lower bound is necessary to guarantee the existence of a feasible solution. The upper bound is set at the average number of recorded flight hours per day multiplied by the number of days in the planning horizon.

As the size of the model increases, so too naturally does the time it takes to solve the model. For fleets containing a large number of aircraft, it seems that the model becomes too large to solve with exact methods within a reasonable time. Therefore another solution method will have to be constructed to generate good solutions in a faster way.

In this chapter, a city-day network-based model is presented, which optimizes the maintenance routing and scheduling for a set of aircraft. The objective of the model is to create routes that can handle all the maintenance requirements for aircraft and minimize the weighted lost flying time of the individually considered tasks. The maintenance feasibility of each route concerning short and low-interval tasks is ensured through the use of lines, which are flight routes that can span multiple days and start and/or end at a maintenance station.

4 Solution Approach

Two types of optimization algorithms exist: exact and heuristic algorithms [41]. Exact algorithms guarantee that they will find the optimal solution in a finite amount of time, whereas heuristics do not have this guarantee, typically returning solutions worse than optimal. The guarantee of finding the optimal solution means that exact algorithms both have to find this solution *and* also prove that the solution is optimal. For many real-life optimization problems, exact algorithms are not a suitable solution, as their solving time is too great [41].

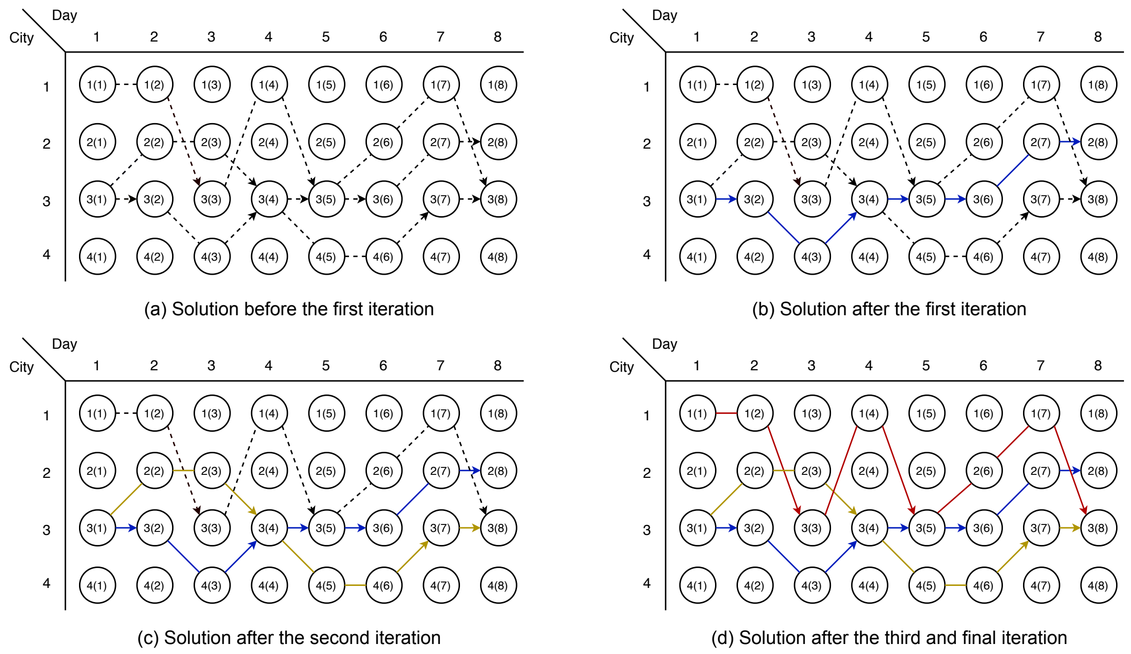
Due to advances in hardware and exact solution methods, several MIP models can reach optimality within a reasonable amount of time or close to it [6]. This has led to a relatively new research area called matheuristics that attempts to combine exact algorithms with heuristics. The resulting method is usually an integration of exact methods solving subproblems in a higher level heuristic [41]. Matheuristics have been applied to several different routing problems. One of the main matheuristic approaches is the decomposition approach [6]. In the decomposition approach, the problem is decomposed into smaller subproblems, that are solved through mathematical programming. A feasible solution for the overall problem is obtained from the solutions of the smaller subproblems. Decomposition approaches are especially suitable for handling complex and interrelated problems [6], which could make it a good fit for the problem presented in chapter 3. The following two distinct types of decompositions have been formulated for our problem:

1. Decomposition by aircraft matheuristics; and
2. Rolling horizon matheuristics

The decomposition by aircraft matheuristics are presented in section 4.1, and section 4.2 presents the rolling horizon matheuristics. In every section, the results of the presented matheuristics are compared with the results of the exact solution method under the same parameter settings. Furthermore, the matheuristics are run for larger problem sets for which there are no optimal solutions available, and these results are presented as well. For the large problem sets, the maintenance capacity was increased to 150 man-hours per night to guarantee the existence of feasible solutions. If the matheuristic was not completed after 1 hour of computation time, the results are left blank.

4.1. Decomposition by Aircraft Matheuristics

In the decomposition by aircraft matheuristic, our problem is decomposed into N subproblems, where N equals the total number of aircraft. Each subproblem generates the route and maintenance schedule for a single tail number by solving the corresponding MIP to optimality using a branch and bound. For the first aircraft, all the lines and the full maintenance capacity are available. After the solution is obtained, the used lines are removed from the network, and the remaining maintenance capacity is updated. Subsequently, the MIP with the remaining lines and maintenance capacity is solved for the next aircraft until all aircraft are assigned a route and maintenance schedule. Figure 4.1 gives a graphical representation of the order in which a solution is calculated using a decomposition by aircraft method. City 3 again represents the maintenance station in the figure. It can be seen that the first aircraft (blue) has all the lines at its disposal, whilst the last aircraft (red) has to make do with whatever lines are left in the network. Two decomposition by aircraft matheuristics have been modeled in this thesis. In section 4.1.1, a depth-first and random search combination is presented, and in section 4.1.2, a successive route checking matheuristic is designed.



(a) Solution before the first iteration

(b) Solution after the first iteration

(c) Solution after the second iteration

(d) Solution after the third and final iteration

Figure 4.1: Visualization of the of the solution process of a decomposition by aircraft method for a set of 3 aircraft and a 7-day planning horizon. City 3 is the maintenance station.

4.1.1. A Depth-first and Random Search Combination

The decomposition by aircraft method presented in this section is inspired by the depth-first and random search heuristic used by Sriram & Haghani [40], which yielded good results for an aircraft maintenance routing problem that is similar to our own. In this matheuristic, a list of all considered aircraft is made in random order, named the aircraft list. The first aircraft on the list is selected, and the best possible route is constructed from all the available lines by solving the mathematical model presented in chapter 3, adapted for a single aircraft.

The available starting lines the aircraft has at its disposal are organized in a feasibility list, that contains the lines that it could feasibly choose as its first line. Lines that, if selected, would create feasibility problems for upcoming aircraft are excluded from the feasibility list and are thus not available for selection, because they need to be reserved for an upcoming aircraft. For example, consider a fleet of three aircraft: aircraft A, aircraft B and aircraft C. Aircraft A has two feasible starting lines: line 1 and line 2, aircraft B has the same two feasible starting lines: line 1 and line 2, and aircraft C has three feasible starting lines: line 1, line 2 and line 3. If aircraft C is the first appearing aircraft on the aircraft list, it should not be able to select line 1 or line 2 as its starting lines, because this will create an infeasible schedule when the final aircraft on the aircraft list is selected. Therefore, line 1 and line 2 would not be included in the feasibility list of aircraft C. Whether or not a line is feasible depends on the starting location of the aircraft and the due date of its first daily check.

After an aircraft route is constructed, the used lines are removed from the network, and the feasibility list and remaining maintenance capacity per day for the maintenance station are updated. Subsequently, the next aircraft on the list is selected, and the process repeats itself until all the aircraft in the list are assigned routes and maintenance schedules. At the end of the process, a feasible solution is found, and all the solution values of the corresponding aircraft are summed to get the weighted value of the total lost flying time. This entire process is executed over multiple iterations, where the aircraft list is randomly shuffled for each iteration. If the found objective value of an iteration is lower than that of the previous best-found solution, the current iteration's solution is saved as the current best solution. At the end of the matheuristic, the best-found solution is presented as the final solution. The flow chart of this solution process is presented in figure 4.2. Table 4.1 presents the results of the depth-first and random search matheuristic and compares them with the results of the exact solution method. In table 4.2, the matheuristic solutions are presented for larger problem sizes with up to 25 aircraft.

Table 4.1: Performance of the depth-first and random search matheuristic with 10 iterations in several problem sets compared to the exact solution method

# AC	# Tasks	CPU time (sec)		Objective value		Optimality gap in %
		Matheuristic	Exact	Matheuristic	Exact	
5	10	7	1	278	258	7.8
	20	8	2	511	471	8.5
6	10	8	1	315	304	3.6
	20	12	7	675	607	11.2
7	10	11	5	467	385	21.3
	20	16	14	802	727	10.3
8	10	12	19	462	424	9.0
	20	20	44	945	863	9.5
9	10	18	53	476	434	9.7
	20	24	110	1043	913	14.2
10	10	24	632	522	432	20.8
	20	48	2138	1102	937	17.6
11	10	28	1269	563	471	19.5
					Average	**12.5**

Table 4.2: Performance of the depth-first and random search matheuristic in several large-sized problems with 10 iterations

# AC	# Tasks	CPU time (sec)	Objective value
15	10	120	797
	20	229	1903
20	10	232	1192
	20	485	2641
25	10	1570	1450
	20	-	-

The results presented in table 4.1, and table 4.2 are found after ten iterations. This means that solutions have been found for ten different orders of the aircraft list. The total possible orders of the aircraft list equals the factorial of the number of aircraft included, which for all test cases vastly surpasses the number of ten. This means that only a small fraction of the solution space is explored. Greatly increasing the number of iterations is not feasible because this proportionally increases the computation time. The explanation behind the long computation times for a single iteration is that the first aircraft on the list of every iteration has a large number of lines to choose from, resulting in a MIP that still requires a substantial amount of time to solve.

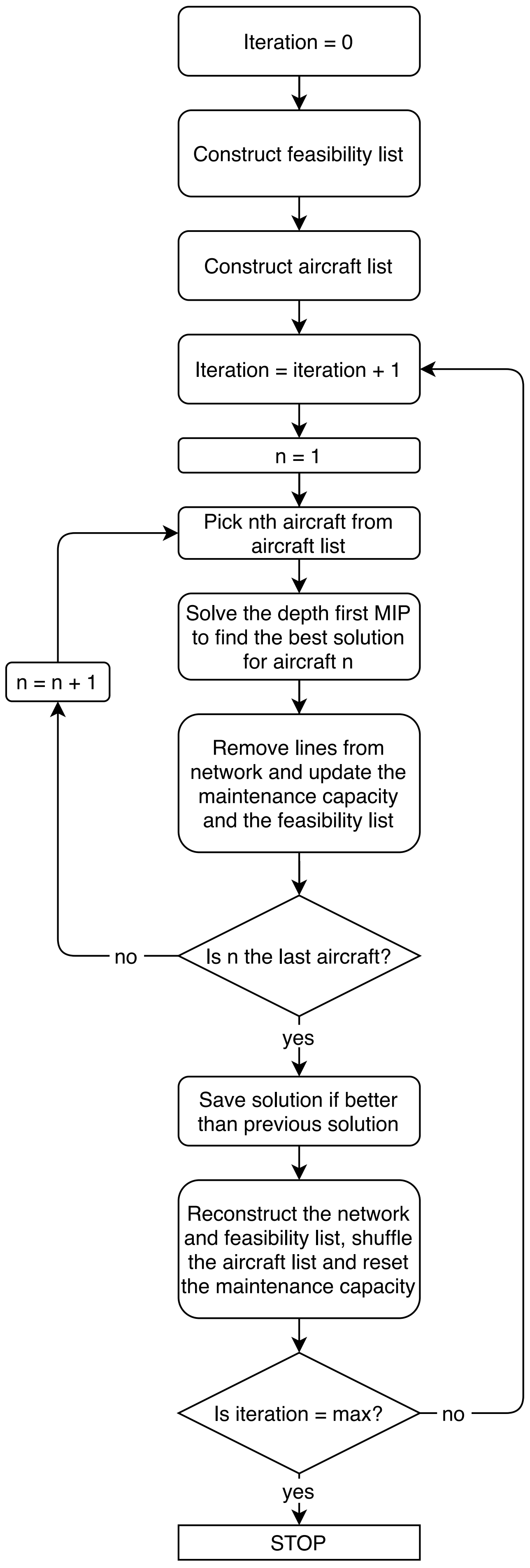

Iteration = 0
Construct feasibility list
Construct aircraft list
Iteration = iteration + 1
n = 1
Pick nth aircraft from aircraft list
Solve the depth first MIP to find the best solution for aircraft n
Remove lines from network and update the maintenance capacity and the feasibility list
Is n the last aircraft?
no
n = n + 1
yes
Save solution if better than previous solution
Reconstruct the network and feasibility list, shuffle the aircraft list and reset the maintenance capacity
Is iteration = max?
no
yes
STOP

Figure 4.2: Flowchart of the depth-first and random search matheuristic

4.1.2. Successive Route Checking Matheuristic

In the successive route checking matheuristic (SRCM), similar to the depth-first and random search matheuristic, aircraft are given their routes one by one. However, instead of constructing routes from the available lines, all the possible routes are created before solving the model. Subsequently, a random aircraft list is constructed in a similar way as in section 4.1.1. For the first aircraft on the list, an enumerative search is done by solving a task-planning MIP for every feasible route. Again, a feasibility list is used that only contains routes that, if selected, do not create feasibility problems for the current and upcoming aircraft. The route corresponding with the best solution score is selected for the first aircraft. Subsequently, the list of remaining possible routes and the remaining maintenance capacities per day are updated. Next, the second aircraft in the list is selected, and the process repeats itself until routes and maintenance schedules are assigned to all the tail numbers. At the end of the process, a feasible solution is found, and all the solution values of the corresponding maintenance schedules are summed to get the weighted value of the total lost flying time. This entire process is executed over multiple iterations, where the aircraft list is randomly shuffled for each iteration. If the found objective value of an iteration is lower than that of the previous best-found solution, the current iteration's solution is saved as the new best solution. At the end of the matheuristic, the best-found solution is presented as the final solution. A flowchart of the successive route checking matheuristic is given in figure 4.3.

Because the route is given as input during each MIP, the routing constraints can be eliminated. Also, since the length of the route is known beforehand, the specific set of tasks that need to be scheduled can be determined. Another advantage is that the quadratic constraints (3.23) can be linearized. In the successive route checking matheuristic the following MIP is solved for each selected route:

Table 4.3: Parameters, indexes, sets and decision variables for SRCM

Parameters & Indexes:	
n_d	number of days in the planning horizon
n_j	number of lines in the route
n_t	number of tasks to be planned
d	index for days d = 1,2,...,n_d
j	index for lines j = 1,2,..,n_j
t	index for tasks t = 1,2,...,n_t
L_j	number of flight hours flown at the end of line j
DD_t	due date of task t
RFH_t	remaining number of legal flight hours of task t at the start of the planning horizon
$Date_j$	end date of line j
S_t	size in man-hours of task t
ms_j	binary value which equals 1 if line j ends on the maintenance station, 0 otherwise
M_d	remaining capacity of the maintenance station in man-hours on day d
Avg	average number of recorded flight hours per day per aircraft
Sets:	
J	set of all lines in the considered route
Je_d	set of all lines ending on day d
T	set of tasks to be planned within the route
Decision Variables:	
y_{tj}	= 1 if maintenance task $t \in T$ is planned after line j, 0 otherwise

$$Minimize \quad \sum_{j \in J} \sum_{t \in T} y_{tj} * S_t * min(Avg * (DD_{\mathrm{i,t}} - Date_{\mathrm{j}}), RFH_{\mathrm{t}} - L_j) \tag{4.1}$$

$$\sum_{j \in J} y_{tj} * ms_j = 1 \qquad \forall t \in T \tag{4.2}$$

$$y_{tj} * Date_j \leq DD_t \qquad \forall j \in J, t \in T \tag{4.3}$$

$$y_{tj} * L_j \leq RFH_t \qquad \forall j \in J, t \in T \tag{4.4}$$

$$\sum_{j \in Je_d} \sum_{t \in T} y_{tj} * S_t \leq M_d \qquad d = 1, \dots, n_d \tag{4.5}$$

$$y_{tj} \in \{0, 1\} \qquad \forall t \in T, j \in J \tag{4.6}$$

The results of the successive route checking matheuristic are presented in tables 4.4 and 4.5. It appears that the speeding up of the matheuristic due to the simplification of the model is nullified by the large number of possible routes, which rises exponentially as the number of aircraft, and thus the number of lines, increases. The total number of possible routes that could be constructed from all the lines in our used data is presented in table 4.6.

Table 4.4: Performance of the successive route checking matheuristic with 10 iterations in several problem sets compared to the exact solution method

# AC	# Tasks	CPU time (sec)		Objective value		Optimality gap in %
		Matheuristic	Exact	Matheuristic	Exact	
5	10	11	1	271	258	5.0
	20	13	2	510	471	8.3
6	10	15	1	336	304	10.5
	20	16	7	652	607	7.4
7	10	21	5	425	385	10.4
	20	26	14	807	727	11.0
8	10	38	19	469	424	10.6
	20	33	44	972	863	12.6
9	10	63	53	481	434	10.8
	20	53	110	1071	913	17.3
10	10	150	632	514	432	19.0
	20	127	2138	1103	937	17.7
11	10	225	1269	572	471	21.4
					Average	**12.5**

Table 4.5: Performance of the successive route checking matheuristic in several large-sized problems with 10 iterations

# AC	# Tasks	CPU time (sec)	Objective value
15	10	594	816
	20	392	1888
20	10	2210	1167
	20	1404	2729
25	10	-	-
	20	-	-

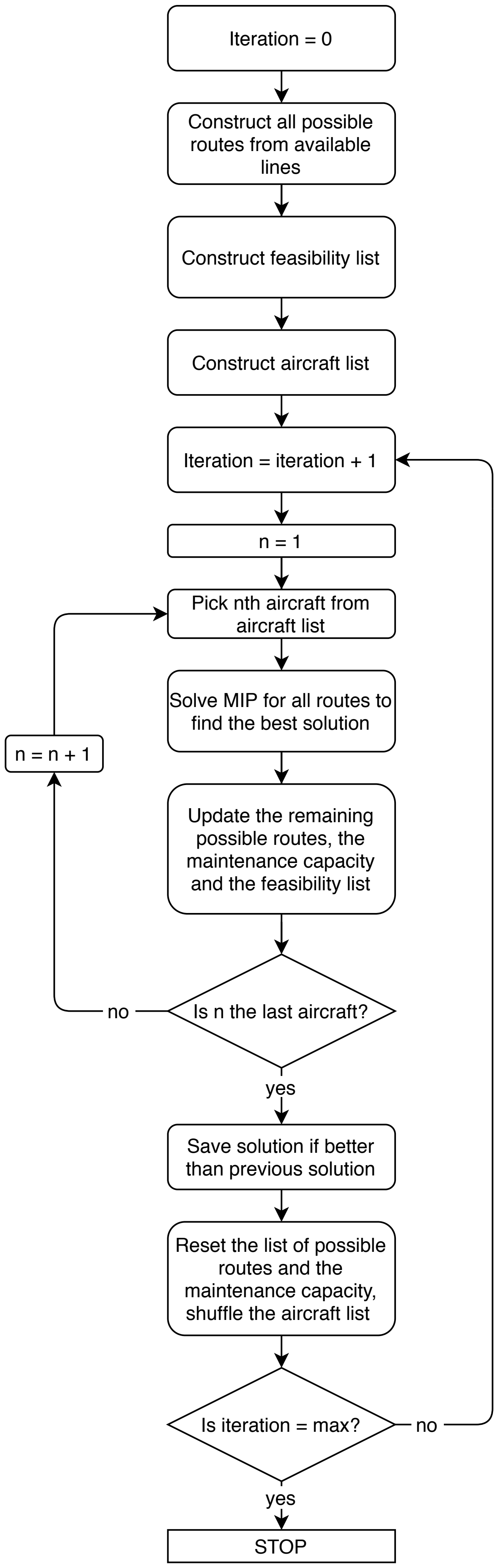

Iteration = 0
Construct all possible routes from available lines
Construct feasibility list
Construct aircraft list
Iteration = iteration + 1
n = 1
Pick nth aircraft from aircraft list
Solve MIP for all routes to find the best solution
Update the remaining possible routes, the maintenance capacity and the feasibility list
Is n the last aircraft?
no
n = n + 1
yes
Save solution if better than previous solution
Reset the list of possible routes and the maintenance capacity, shuffle the aircraft list
Is iteration = max?
no
yes
STOP

Figure 4.3: Flowchart of the successive route checking matheuristic

Table 4.6: Total number of possible routes per set of considered aircraft for a time horizon of seven days

# AC	total number of lines	total number of possible routes
5	19	90
10	37	1576
15	54	5685
20	70	16,921
25	87	49,244

4.2. Rolling Horizon Matheuristics

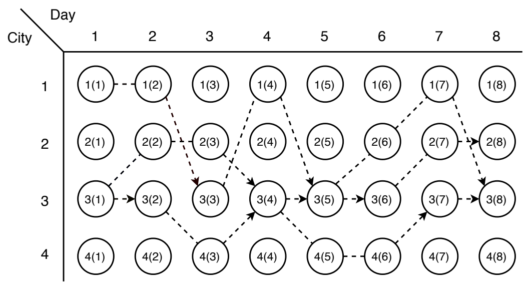
The matheuristic approach of decomposing a problem into time periods is classified as a rolling horizon matheuristic (RHM) by Archetti & Speranza [6]. A rolling horizon matheuristic is a solution methodology to problems in which decisions must be made over time. In a rolling horizon matheuristic, a MIP is repeatedly solved for shorter sub-horizons of the planning horizon, using a branch-and-bound approach [34]. Rolling horizon matheuristics have been applied for several transportation problems, such as Agra et al. [4], who use a rolling horizon matheuristic to solve an inventory routing problem in which ships are routed and scheduled between ports such that the demand for various fuel oil products is satisfied during the planning horizon. Rakke et al. [34] use a rolling horizon matheuristic to create a liquefied natural gas annual delivery program.

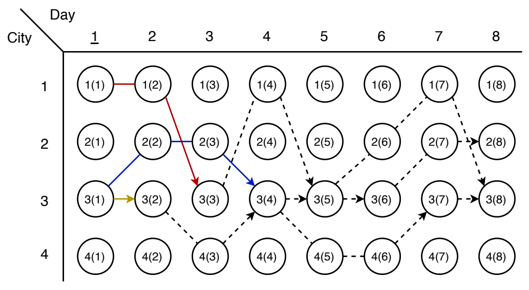
At the beginning of our RHM, all the tail numbers and all the lines that start from day 1 are considered. During the first iteration, a MIP is solved that distributes the starting lines as efficiently as possible and plans any required maintenance within the time period, with the goal of minimizing the total number of lost flying time. After the MIP is solved, the routing variables are frozen, and the sub-horizon shifts forwards. During the second iteration, all the tail numbers and all the lines that start on day 2 are considered. For the tail numbers, this comes down to the aircraft that were assigned 1-day long lines during the previous iteration. The pseudo-code is presented in algorithm 1, and figure 4.4 gives a graphical representation of the rolling horizon matheuristic. Each color in figure 4.4 represents a distinct route for a tail number, and city three is, once again, the maintenance station.

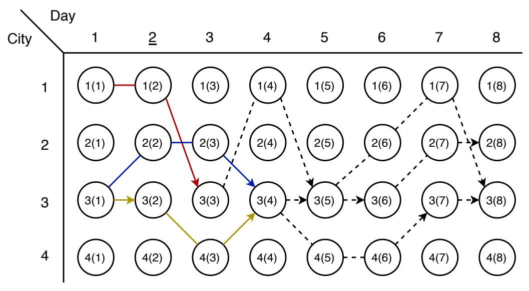
Algorithm 1: Rolling Horizon Matheuristic

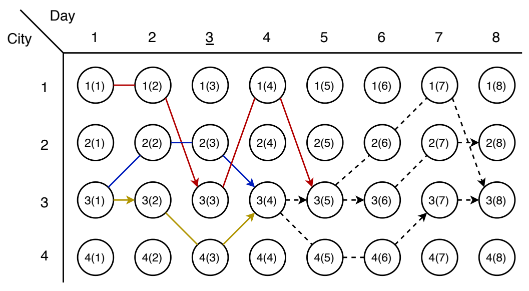
```
k = 0;
U = Number of days in the planning horizon;
while $k = k + 1 \leq U$ do
    1: Select the set of tail numbers and lines starting from day k;
    2: Determine the task set for each tail number that requires consideration;
    3: Solve the MIP, including feasibility constraint if k = 1;
    4: Freeze variables $x_{ij}$ from the current iteration;
end
```

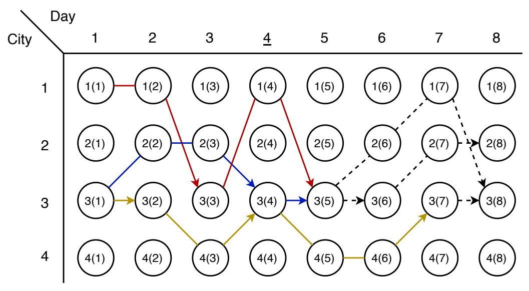
The results of the RHM are presented in table 4.7 and are compared to the results produced by the exact solution method. For most problem sets, the RHM produces much better solutions than the decomposition by aircraft methods, which were presented in section 4.1. The time it takes to solve the given problems is also drastically reduced. The RHM has also been run for the larger sized problem sets, and the results of these tests are presented in table 4.8. Even sets containing 25 aircraft were solved in under 30 seconds, while also presenting better objective values than both the depth-first random search, and the successive route checking matheuristic.

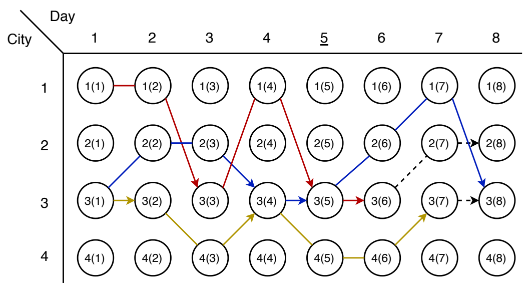

(a) RHM before the first iteration

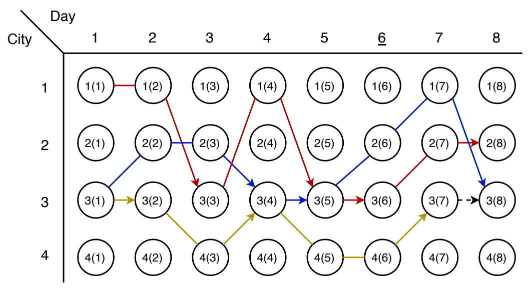

(b) RHM results after first iteration


(c) RHM results after second iteration


(d) RHM results after third iteration


(e) RHM results after fourth iteration


(f) RHM results after fifth iteration


(g) RHM results after sixth iteration

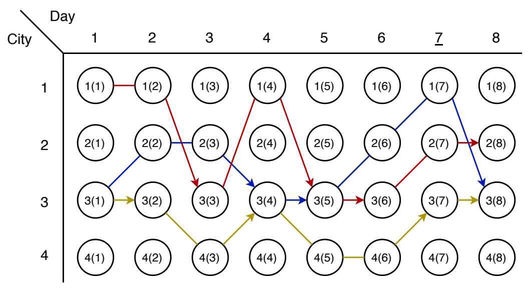

(h) RHM results after seventh iteration

Figure 4.4: Visualization of the functioning of the RHM for a set of 3 aircraft and a 7-day planning horizon. City 3 is the maintenance station.

Table 4.7: Performance of the RHM in several problem sets compared to the exact solution method

# AC	# Tasks	CPU time (sec)		Objective value		Optimality gap in %
		Matheuristic	Exact	Matheuristic	Exact	
5	10	1	1	266	258	3.1
	20	1	2	487	471	3.4
6	10	1	1	308	304	1.3
	20	2	7	607	607	0.0
7	10	1	5	385	385	0.0
	20	2	14	731	727	0.6
8	10	1	19	436	424	2.8
	20	2	44	868	863	0.6
9	10	2	53	446	434	2.8
	20	3	110	939	913	2.8
10	10	2	632	481	432	11.3
	20	3	2138	997	937	6.4
11	10	2	1269	620	471	31.6
					Average	**5.1**

Table 4.8: Performance of the RHM in several large sized problems

# AC	# Tasks	CPU time (sec)	Objective value
15	10	3	746
	20	6	1692
20	10	5	942
	20	11	2280
25	10	8	1200
	20	22	3030

The downside of an RHM, as implemented above, is that it, in a sense, is myopic. It only considers lines that start on the day of the iteration and does not take future lines that appear in an upcoming iteration into account. This can sometimes result in poor solutions, as is the case for the problem set consisting of 11 airplanes and 10 possible tasks for each in table 4.7. To counter this shortsightedness, Rakke et al. [34] and Agra et al. [4] included a forecasting section in their rolling horizon matheuristic. Rakke et al. [34] and Agra et al. [4] decompose a sub-horizon into two time periods: the central period (CP) and the forecasting period (FP). In the central period, the MIP is solved, and the selected routes within it are frozen before moving to the next iteration. The forecasting period provides information to the central period about a larger part of the planning horizon. The idea is that by using a forecasting period, clearly sub-optimal solutions outside of the central period can be avoided so that the myopic nature of a rolling horizon matheuristic is mitigated [34]. Figure 4.5 gives a graphical representation of the sub-horizons used by Agra et al. [4]. To provide the central period with information, a MIP is solved within the forecasting period, which often is a simplified version of the MIP in the central period. Two main elements need to be considered for the forecasting period [34]:

1. A simplification strategy; and
2. The length of the forecasting period

A rolling horizon matheuristic that includes a forecasting section carries certain similarities with a well-established family of control techniques known as Model Predictive Control (MPC) or Receding Horizon Control [9]. MPC is a model-based control approach, where a constrained optimization problem is

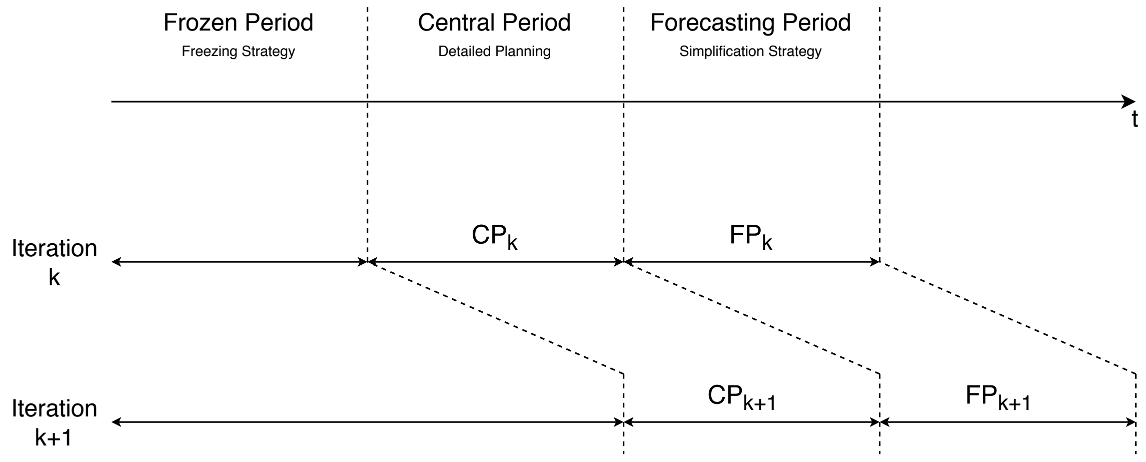


Figure 4.5: The different time-windows within the rolling horizon matheuristic using a forecasting approach [4]

solved over a given forecast horizon to determine a control action sequence for the controller. During every control step, only the first element of this action sequence is carried out. At the next time step, the constrained optimization problem is reformulated and solved again, leading to a new control action sequence of which, again, only the first element is carried out, resulting in a receding horizon strategy. To reduce the computational burden of an MPC, Tian et al. [42] have proposed an Adaptive Control Resolution (ACR) approach. An ACR approach reduces the number of control variables by dividing their problem horizon in a number of phases with decreasing resolution as the phases become more distant in the future. It can be understood that the presented RHM with a forecasting section operates in a similar manner. During every iteration, only the first action (the flight schedule produced in the central period) is frozen or ”carried out”. Implementing a simplification strategy within the forecasting period is similar to implementing the ACR within the MPC, as it reduced the number of variables and is aimed to reduce the computation times.

Sections 4.2.1 and 4.2.2 discuss two different forecasting strategies and present their results.

4.2.1. Exact Forecasting

Using an exact forecasting strategy means that there is no simplification for the forecasting period. The model in the forecasting period is subjected to the same constraints and objective function as in the central period. This means that the full model is solved for a time period that equals the length of the central period plus the length of the forecasting period, but only the lines that depart in the central period are frozen for the next iteration. It can logically be concluded that, if the length of the central period and the forecasting period together equals the length of the original problem’s planning horizon, the optimal solution is found. Algorithm 1 is updated to include the exact forecasting section and is presented in algorithm 2.

It is no longer sufficient only to consider lines and aircraft departing from the maintenance station at the start of day *k*, but instead lines starting within the FP_k must also be considered. These lines need to be assigned either to aircraft that are operating previously frozen lines during the CP_k, or to aircraft that are assigned a new line at the start of the CP_k, which will finish before the end of the FP_k, and thus will need to be assigned a second line within the considered time period. To allow for this, the flow constraints (3.19), have been adapted to the constraints presented in (4.7).

$$x_{ij} - \sum_{j \in C_j} x_{ij} - \sum_{p \in P_j} z_{i,p} \leq 0 \qquad \forall i \in I, j \in Jm \tag{4.7}$$

Table 4.9: Extra parameters and sets used in RHM with exact forecasting

Parameters:	
$z_{i,p}$	binary value which equals 1 if aircraft *i* has flown frozen line *p*, 0 otherwise
Sets:	
Jm	set of all lines apart from the lines starting on day *k*
C_j	set of all lines within CP_k and FP_k that connect to line *j*
P_j	set of all frozen lines that connect to line *j*

Algorithm 2: Rolling Horizon Matheuristic With Exact Forecasting

```
k = 0;
U = Number of days in the planning horizon;
while $k = k + 1 \leq U$ do
    1: Identify the set of lines that start during the $CP_k$ and $FP_k$;
    2: Identify the set of tail numbers that will start a new line during the $CP_k$ and $FP_k$;
    3: Determine the task set for each tail number that requires consideration;
    4: Solve the MIP for the problem defined by $CP_k$ and $FP_k$;
    5: Freeze variables $x_{ij}$ in the central period $CP_k$;
end
```

The RHM with exact forecasting has been run for a forecasting period of both 1 day and 2 days. Increasing the forecasting period further increased the computational burden too much for the larger problem sets. Tables 4.10 and 4.11 present the results for both forecasting lengths for the smaller problem sets. By using an exact forecasting of both 1 and 2 days, the average optimality gap has decreased to 2.7 and 1.5 percent, respectively. The poor result of the RHM without forecasting on the problem set of 11 aircraft and 10 tasks has decreased from 31.6 percent to 11.3 percent for 1-day of forecasting and 3.2 percent for 2-days of forecasting. It can thus be noted that the inclusion of a forecasting section within the rolling horizon matheuristic can substantially improve the objective values.

Table 4.10: Performance of the RHM with 1-day exact forecasting in several problem sets compared to the exact solution method

# AC	# Tasks	CPU time (sec)		Objective value		Optimality gap in %
		Matheuristic	Exact	Matheuristic	Exact	
5	10	2	1	267	258	3.5
	20	2	2	471	471	0.0
6	10	2	1	308	304	1.3
	20	4	7	607	607	0.0
7	10	2	5	393	385	2.1
	20	5	14	741	727	1.9
8	10	5	19	435	424	2.6
	20	7	44	867	863	0.5
9	10	7	53	444	434	2.3
	20	13	110	958	913	4.9
10	10	16	632	432	432	0.0
	20	26	2138	965	937	3.0
11	10	18	1269	534	471	13.4
					Average	**2.7**

Table 4.11: Performance of the RHM with 2-day exact forecasting in several problem sets compared to the exact solution method

# AC	# Tasks	CPU time (sec)		Objective value		Optimality gap in %
		Matheuristic	Exact	Matheuristic	Exact	
5	10	2	1	266	258	3.1
	20	4	2	471	471	0.0
6	10	4	1	308	304	1.3
	20	7	7	607	607	0.0
7	10	5	5	385	385	0.0
	20	8	14	727	727	0.0
8	10	10	19	430	424	1.4
	20	19	44	867	863	0.5
9	10	25	53	449	434	3.5
	20	53	110	939	913	2.8
10	10	46	632	437	432	1.2
	20	250	2138	962	937	2.7
11	10	62	1269	486	471	3.2
					Average	**1.5**

Tables 4.12 and 4.13 present the results for larger problem sets. When the length of the forecasting period equals one day, the problem can be solved within reasonable time for sets up to 25 aircraft, with a possible 20 tasks for each aircraft. If the forecasting period is increased from one to two days, the computation times are greatly increased however. The problem set containing 20 aircraft and 10 tasks for each aircraft could not be completed within one hour.

Table 4.12: Performance of the RHM with 1-day exact forecasting in several large sized problems

# AC	# Tasks	CPU time (sec)	Objective value
15	10	45	671
	20	131	1660
20	10	157	927
	20	513	2224
25	10	413	1149
	20	825	2898

Table 4.13: Performance of the RHM with 2-day exact forecasting in several large sized problems

# AC	# Tasks	CPU time (sec)	Objective value
15	10	304	643
	20	583	1547
20	10	-	-
	20	-	-

4.2.2. Average Flight Hour Forecasting

In this section, a strategy is presented that solves a simplified model for the forecasting period. As a first simplification, the capacity constraints of the maintenance station (equation 3.26) are removed. As a second simplification, the model will not actually plan the individual maintenance tasks within the forecasting period, but it will make a prediction of future costs if an aircraft is appointed to operate a future line. The overall objective function will then comprise of a part that represents the *actual* lost flying time that is incurred during the central period, and the *predicted* lost flying time that is expected to be incurred in the future. To predict the future costs of a certain aircraft/line combination the average number of flight hours per day (given as an input value) is used. In this way the number of recorded flight hours for each tail number at the start and end of each line within the forecasting period are predicted. Based on the number of tasks that are due within the forecasting period (or expected to be due), the model will assign a predicted tail-number dependent cost for each forecasted line. This is done by calculating the lost flying time that would be incurred due to maintenance tasks having to be planned early if this line is flown.

The term presented in equation 4.8 is added to objective function 3.15 and introduces a new decision variable: q_{if}, which equals 1 if a line within the forecasting period is assigned to aircraft *i*. Furthermore, constraints 4.9 and 4.10 are added to the model. A separate decision variable is introduced because, in contrast to the RHM with exact forecasting, the central period and the forecasting period are no longer subject to the same set of constraints. The pseudo-code of the RHM with average flight hour forecasting is given in algorithm 3.

Table 4.14: Extra parameters, sets and decision variables used in RHM with average flight hour forecasting

Parameters:	
$Cost_{i,f}$	predicted cost if future line *f* is assigned to aircraft *i*
$z_{i,p}$	binary value which equals 1 if aircraft *i* has flown frozen line *p*, 0 otherwise
Sets:	
F	set of all lines starting in the forecasting period
C_f	set of all lines, starting in the central period, that connect to line *f*
P_f	set of all frozen lines that connect to line *f*
R_f	set of all lines, starting in the forecasting period, that connect to line *f*
Decision Variables:	
q_{if}	= 1 if forecasted line *f* is assigned to aircraft *i*, 0 otherwise

$$Minimize \quad \sum_{i\in I}\sum_{f\in F} q_{if} * Cost_{i,f} \tag{4.8}$$

$$\sum_{i\in I} q_{if} = 1 \qquad \forall f \in F \tag{4.9}$$

$$q_{if} - \sum_{j\in C_f} x_{ij} - \sum_{p\in P_f} z_{i,p} - \sum_{f\in R_f} q_{if} \leq 0 \qquad \forall i \in I, f \in F \tag{4.10}$$

Algorithm 3: Rolling Horizon Matheuristic With Average Flight Hour Forecasting

```
k = 0;
U = Number of days in the planning horizon;
while $k = k + 1 \leq U$ do
    1: Identify the set of lines that start during the $CP_k$ and $FP_k$;
    2: Identify the set of tail numbers that will start a new line during the $CP_k$ and $FP_k$;
    3: Determine the task set for each tail number that requires consideration;
    4: Calculate the predicted costs of the lines starting in the $FP_k$;
    5: Solve the MIP defined by $CP_k$ and $FP_k$;
    6: Freeze variables $x_{ij}$ in the central period $CP_k$;
end
```

The RHM with average flight hour forecasting has been tested for multiple lengths of the forecasting period, ranging from 1 day to 6 days. The results of the matheuristic for a forecasting period of 1 day, are presented in tables 4.15 and 4.16. The average optimality gap of the RHM has decreased from 5.1% to 2.9% by including a 1-day forecast based on average flight hours. Furthermore, the required computation times for the matheuristic stay below 35 seconds, even for problem sets containing 25 aircraft with a possible 20 tasks for each.

Table 4.15: Performance of the RHM with 1-day average flight hour forecasting in several problem sets compared to the exact solution method

# AC	# Tasks	CPU time (sec)		Objective value		Optimality gap in %
		Matheuristic	Exact	Matheuristic	Exact	
5	10	1	1	267	258	3.5
	20	1	2	471	471	0.0
6	10	1	1	308	304	1.3
	20	2	7	607	607	0.0
7	10	1	5	385	385	0.0
	20	3	14	731	727	0.6
8	10	2	19	427	424	0.7
	20	3	44	868	863	0.6
9	10	2	53	458	434	5.5
	20	4	110	950	913	4.1
10	10	3	632	432	432	0.0
	20	4	2138	977	937	4.3
11	10	4	1269	553	471	17.4
					Average	**2.9**

Table 4.16: Performance of the RHM with 1-day average flight hour forecasting in several large sized problems

# AC	# Tasks	CPU time (sec)	Objective value
15	10	6	665
	20	9	1715
20	10	10	931
	20	17	2278
25	10	15	1223
	20	31	2992

Table 4.17 presents a summary of the results for problem sets up to 11 aircraft for all the matheuristics that have been presented in this chapter. The results of the RHM, using average flight hour forecasting, with a forecasting period longer than one day, are included in this table. The average optimality gap in percent represents how close the solutions of the given matheuristic were on average to the results produced by the exact solution method. The longest recorded time refers to the longest computing time, in seconds, the matheuristic required to complete, out of all the tests for up to 11 aircraft and 10 tasks for each. From this data, it appears that the rolling horizon matheuristics all present solutions with superior averaged objective values to the decomposition by aircraft methods. The RHMs with exact forecasting present the best solutions, but the required computation time rises steeply when increasing the forecasting period. It also appears that the solution results of the RHMs with average

flight hour forecasting do not consistently improve when increasing the forecasting period. An explanation for this might be that the decrease in prediction accuracy offsets the benefit of longer forecasting.

Table 4.18 presents a summary of the results for larger problem sets of up to 25 aircraft for all the presented matheuristics. In the last row of the table, the average gap with the best found objective value is presented for each matheuristic. Because the exact solution method is not able to solve for such large problem instances, this gap does not equal the optimality gap but instead the gap with the best found objective value from all matheuristics. Judging by this average gap, it seems clear that again the rolling horizon matheuristics outperform the decomposition by aircraft matheuristics in terms of objective values. Similar to the results of the smaller problem sets, the RHM with exact forecasting generates the best objective values for larger sets. However, depending on the length of the forecasting period, its computation time becomes too hefty for the largest problem sizes. The improvement in objective values in the smaller problem sets, generated by augmenting the RHM with the forecasting section based on average flight hours, seems to be reduced for larger sets. A forecasting length of two days, in fact, produced worse results than the RHM without forecasting. Also for the larger problem sets there seems to be no clear improving trend when increasing the forecasting period for the RHM with average flight hour forecasting.

Table 4.17: Comparison of the average optimality gap and the longest recorded solving time per matheuristic of all problem sets for which an exact could be found.

	DFRS	SRCM	RHM NF	RHM EF-1	RHM EF-2	RHM AF-1	RHM AF-2	RHM AF-3	RHM AF-6
Average Gap in %	12.5	12.5	5.1	2.7	1.5	2.9	3.6	3.6	3.6
Longest Recorded time in s	48	225	3	26	250	4	6	8	10

DFRS: Depth-First Random Search, SRCM: Successive Route Checking Matheuristic, RHM NF: Rolling Horizon Matheuristic - No Forecasting, EF-1: Exact Forecasting For 1 Day, AF-1: Average Flight Hour Forecasting for 1 day.

Table 4.18: Comparison of the objective values and solving times of the discussed matheuristics for large problem sets

		DFRS	SRCM	RHM NF	RHM EF-1	RHM EF-2	RHM AF-1	RHM AF-2	RHM AF-3	RHM AF-6
15 AC, 10 tasks	Obj. value	797	816	746	671	643	665	765	731	717
	CPU time (s)	120	594	3	45	304	6	9	10	13
15 AC, 20 tasks	Obj. value	1903	1888	1692	1660	1547	1715	1640	1602	1616
	CPU time (s)	229	392	6	131	583	9	13	14	17
20 AC, 10 tasks	Obj. value	1192	1167	942	927	-	931	974	948	945
	CPU time (s)	232	2210	5	157	-	10	15	18	24
20 AC, 20 tasks	Obj. value	2641	2729	2280	2224	-	2278	2309	2215	2296
	CPU time (s)	485	1404	11	513	-	17	22	28	30
25 AC, 10 tasks	Obj. value	1450	-	1200	1149	-	1223	1184	1259	1187
	CPU time (s)	1570	-	8	413	-	15	22	35	36
25 AC, 20 tasks	Obj. value	-	-	3030	2898	-	2992	2973	2911	2865
	CPU time (s)	-	-	22	825	-	31	41	54	52
Average gap with best-found score		**24.2%**	**24.5%**	**6.7%**	**2.2%**	**0.0%**	**4.7%**	**6.9%**	**5.1%**	**4.1%**

DFRS: Depth-First Random Search, SRCM: Successive Route Checking Matheuristic, RHM NF: Rolling Horizon Matheuristic - No Forecasting, EF-1: Exact Forecasting For 1 Day, AF-1: Average Flight Hour Forecasting for 1 day.

All the matheuristics have been run for a second data set to verify the results. New flight data was produced, and new task data has been generated. A summary of the results for the smaller problem sets is presented in table 4.19. The results for larger problem sets are presented in table 4.20. Also for the second data set, the rolling horizon matheuristics outperform the decomposition by aircraft matheuristics. The RHM with exact forecasting again produces the best results in terms of objective values. The first notable difference is the decrease in computation times for the second data set. One of the reasons for this disparity could be that the second data set had slightly fewer lines as input flight-data. A second reason could be that the data, in some cases, had some clearly best solutions, which could reduce the problem complexity. The second notable difference is the divergence between the results produced by the RHM without forecasting and the RHM with average flight hour forecasting. The improvement produced by the forecasting is much more substantial than it was with the first data set.

Table 4.19: Comparison of the average optimality gap and the longest recorded solving time per matheuristic of all problem sets for which an exact could be found. These results are derived from a different flight- and task-data set than in table 4.17.

	DFRS	SRCM	RHM NF	RHM EF-1	RHM EF-2	RHM AF-1	RHM AF-2	RHM AF-3	RHM AF-6
Average Gap in %	8.6	9.6	3.6	1.1	0.55	0.69	0.83	1.63	1.18
Longest Recorded time in s	37	169	3	13	41	4	5	7	8

DFRS: Depth-First Random Search, SRCM: Successive Route Checking Matheuristic, RHM NF: Rolling Horizon Matheuristic - No Forecasting, EF-1: Exact Forecasting For 1 Day, AF-1: Average Flight Hour Forecasting for 1 day.

Table 4.20: Comparison of the objective values and solving times of the discussed matheuristics for large problem sets. These results are derived from a different flight- and task-data set than in table 4.18.

		DFRS	SRCM	RHM NF	RHM EF-1	RHM EF-2	RHM AF-1	RHM AF-2	RHM AF-3	RHM AF-6
15 AC, 10 tasks	Obj. value	1050	968	920	868	877	870	899	918	891
	CPU time (s)	59	415	3	26	113	6	8	10	14
15 AC, 20 tasks	Obj. value	2493	2532	2299	2240	2179	2221	2191	2191	2240
	CPU time (s)	120	375	6	66	361	8	11	13	17
20 AC, 10 tasks	Obj. value	1349	1486	1225	1191	1130	1199	1148	1165	1156
	CPU time (s)	265	1649	6	117	411	9	14	18	23
20 AC, 20 tasks	Obj. value	3376	3291	2961	2887	2780	2906	2843	2940	2855
	CPU time (s)	321	1493	10	302	1046	15	22	24	29
25 AC, 10 tasks	Obj. value	1687	-	1511	1273	-	1347	1421	1328	1386
	CPU time (s)	512	-	8	260	-	14	22	26	35
25 AC, 20 tasks	Obj. value	4174	-	3519	3467	-	3580	3505	3505	3555
	CPU time (s)	1626	-	22	588	-	27	35	40	53
Average gap with best-found score		**21.5%**	**19.4%**	**7.8%**	**2.0%**	**0.3%**	**3.6%**	**3.5%**	**3.4%**	**3.6%**

DFRS: Depth-First Random Search, SRCM: Successive Route Checking Matheuristic, RHM NF: Rolling Horizon Matheuristic - No Forecasting, EF-1: Exact Forecasting For 1 Day, AF-1: Average Flight Hour Forecasting for 1 day.

4.3. Discussion of the Solution Approaches

Due to advances in hardware and exact solution methods, the usability of exact solution methods has increased within the domain of heuristics. These advances have led to a relatively new research area called matheuristics, where exact solution methods can be integrated into heuristic frameworks. In this chapter, several matheuristics have been designed that can be grouped into two main categories:

decomposition by aircraft matheuristics and rolling horizon matheuristics. We found that rolling horizon matheuristics consistently outperformed their decomposition by aircraft counterparts and that they generated good solutions in quick fashion. To combat the shortsightedness that any rolling horizon heuristic faces, the rolling horizon method can be augmented with a forecasting section that provides information about a larger part of the planning horizon. Equipping the rolling horizon matheuristics with an exact forecasting section decidedly improves the obtained solutions in terms of objective value even further. Depending on the size of the fleet, however, using exact forecasting for a number of days may or may not be feasible due to the accompanying computation time increase. In these cases, it may be more beneficial to employ a forecasting strategy based on average flight hours. The forecasting strategy based on average flight hours produces good solutions in a very quick fashion. The effectiveness of this forecasting strategy inherently depends on the degree of consistency of day-routes within an airline's fleet in terms of flight hours. If the variability of the flown flight hours per day is within bounds, the rolling horizon matheuristic using an average flight hour forecasting could potentially produce excellent results.

5 Experimental Study

In this chapter an experimental study will be performed with the aim of analyzing the potential impact of an individual task-based maintenance planning, as produced by one of the presented matheuristics, on the airline industry. The results of the schedules created by the matheuristic will be compared to a standard letter check schedule based on two key performance indicators: the total number of days an aircraft has to be taken out of operations and the incurred amount of lost flying time due to tasks being executed before they are due. Furthermore, the distribution of required man-hours at the maintenance station per night will be compared. For this experimental study, the rolling horizon matheuristic with one day exact forecasting is selected as it has proven to produce good results in reasonable time for similar problem sizes that will be used in this experimental study. For the sake of simplicity, only A-checks, and the individual maintenance tasks within them, have been considered in this experimental study.

The extent of the improvements of the use of an individual task-based maintenance program has already been shown by Senturk & Ozkol [38, 39]. In their work, they showed that the ground downtime for the A-checks can be reduced to zero days if the individual tasks in the checks are planned individually inside overnight checks. In this experimental study, we are going to check whether one of the presented matheuristics is capable of matching this performance. Furthermore, we are going to compare the number of lost flying hours experienced between our selected matheuristic and the use of a standard A-check planning. To successfully do this, we will make a number of assumptions about the current state of the A-check planning. Subsequently, we will judge the quality of the letter-check planning strategy based both on the number of days that aircraft have been taken out of operations and the total number of lost flying hours that have incurred. We will then execute one of the presented matheuristics, within the same given framework, and judge its performance on the same criteria.

In practice, it is likely that the routing and planning for the fleet would be updated at the end of each day to produce the new current schedule for the considered planning horizon. To mimic this, the selected matheuristic will be transformed into a model with a running horizon. The model will be run before the start of every day for 60 days with a planning horizon of 7 days, and only the solution of the first day will be saved as the actually carried out flight and maintenance schedule. Furthermore, the capacity of the maintenance station has been set at 75 man-hours per night.

For this experimental study, a fleet of 15 Boeing 737s of a large European airline has been considered. The flight data of these aircraft is extracted from a paid online aircraft-tracking service. Any gaps in the flight data are supplemented with fabricated data in line with the rest of the flight data. The used flight data runs from October 1st, 2019 to November 30th, 2019, spanning 60 days. Furthermore, the interval of an A-check is put at 60 days. The check interval of 60 days has been selected so that a clearly bounded time period can be chosen for the running horizon, in which each tail number is guaranteed to have one A-check. Kinnison and Sidiqi [20] set the interval for an average A-check to be 600 flight hours (see table 2.1), which for most European continental fleets would, in practice, not differ too much from an interval of 60 days. The A-checks are spaced as evenly as possible in the 60-day period. This is what would be expected in practice as well, because having all aircraft undergo base maintenance in, for example, 1 week would severely disrupt an airline's ability to operate its published flight schedule. The distribution of planned A-checks over the given interval of 60 days is visualized in figure 5.1. In this case, every four days a tail number within the fleet will undergo an A-check.

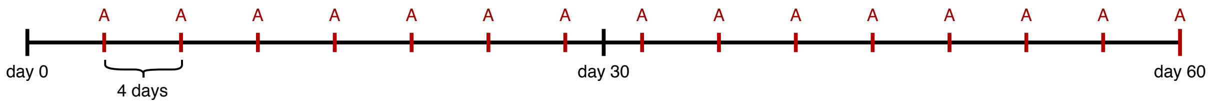


Figure 5.1: A visualization of the distribution of the scheduled A-checks for a fleet of 15 aircraft and a check interval of 60 days.

As discussed in section 2.1.2, we consider an A-check to contain 100 individual maintenance tasks, with an average length of 1 man-hour per task. In our experimental study, the assumption has been made that half the tasks are part of a so-called 'core set' of tasks, that are recurring in every A-check. These core tasks are given coincident due dates with the planned A-check. In practice, this might give too optimistic a view of the standard A-check planning, as it means that all the tasks in the core set will have a combined total of zero lost flying hours. It is, however, probably safer to overestimate the efficiency of the current state of the industry than to underestimate it when evaluating a new operational strategy. The size in man-hours of each core task is generated following a uniform distribution where the lower bound equals 0.5 man-hours and the upper bound equals 1.5 man-hours. The set of core tasks is the same for each aircraft in the considered fleet, only the given due date of the set differs per aircraft.

The tasks that are not part of the core set represent tasks that just happen to be due within the 60-day interval between two consecutive A-checks, and are moved forward to be placed inside the first of these checks. For each aircraft, a unique set of 'non-core' tasks is generated. For each non-core task, the size in man-hours is generated in the same way as for the tasks in the core set. Furthermore, at the start of the interval, each non-core task is assigned a number of legal remaining flight hours the aircraft can operate before the task is due. The remaining number of legal flight hours are generated following a uniform distribution as well. However, putting the lower bound of this distribution at 0 flight hours would lead to an infeasible start of the model, because a task would be due before the aircraft has had a chance to visit a maintenance station for the first time. Therefore, the lower bound is set at a specific number of flight hours, after which the aircraft is guaranteed to have had the ability to visit a maintenance station. By examining the given lines at the start of the model, this number can be determined. The upper bound of the uniform distribution is determined by multiplying the number of average flight hours per day by 60. A visual example of a possible distribution of several due times of non-core set tasks for a single aircraft is given in figure 5.2. In this image, each diamond represents a due date of a non-core task. In a standard letter-check planning strategy, all the tasks would be moved forward to be executed in the marked A-check. The tasks that are due before the marked A-check would have been moved up to an A-check occurring before day 0. Because the non-core tasks are given a remaining number of legal flight hours at day 0, it is actually unknown at which exact day they will be due, therefore figure 5.2 might not be 100 percent accurate and is more for illustrative purposes. The distribution of core tasks and non-core tasks inside the A-check is set to fifty-fifty. This assumption was made after discussion with managers and engineers at a large European MRO providing company, who confirmed that it is not unrealistic.

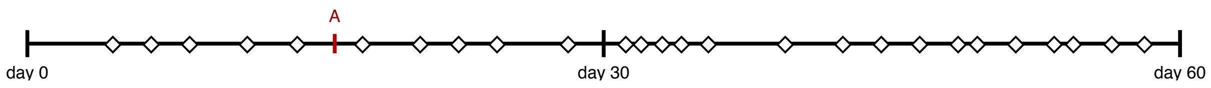


Figure 5.2: Visual example of possible due moments of non-core tasks, where each non-core task is represented by a diamond.

The selected matheuristic, whose solution will be compared to the standard A-check planning, is the rolling horizon matheuristic with one day of exact forecasting. This matheuristic has been selected for the experimental study because it has proven to provide good solutions and reasonable solving times for problem sets of this size.

5.1. Results of the Experimental Study

This section will present the results of the experimental study. It must be noted that the presented lost flying hours are actually the *weighted* lost flying hours. Section 3.1.1 explains that the size of the task is taken as a weight factor because it is more beneficial to fully utilize the task interval of larger tasks than smaller ones. Since the average size of a task equals one man-hour, and the main interest is to examine the differences between the two methods, not the absolute sizes, we present it as the lost flying hours.

Figure 5.3 and table 5.1 present the results on the lost flying days incurred during the experimental study for both tested approaches. The A-check planning requires every aircraft to be taken out of operations for a single day to complete the check. This means that for a fleet of 15 aircraft, a total of 15 lost flying days were incurred over the given interval. As shown in figure 5.1, the A-checks are distributed evenly over the total interval, which can be recognized in figure 5.3 by the regularly spaced jumps in the value of the accumulated lost flying days.

The proposed matheuristic was able to plan all the required maintenance tasks within overnight ground times at the maintenance station, which resulted in a total of zero lost flying days. The accumulated number of lost flying days incurred when using the matheuristic is represented by the green line in figure 5.3. Because this value remains zero throughout the entire interval, the green line runs over the x-axis.

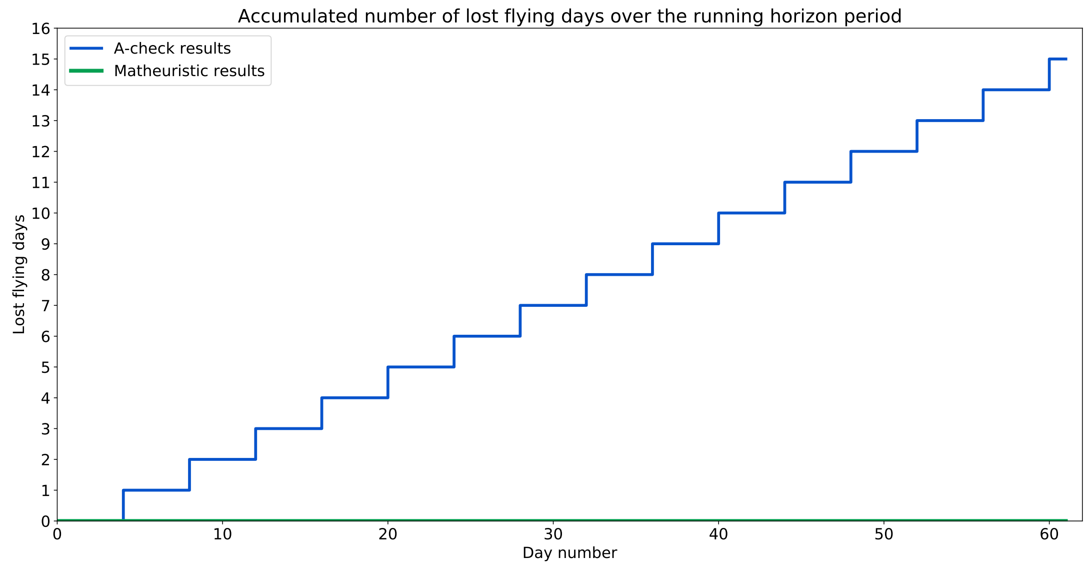


Figure 5.3: Visualization of the accumulation of lost flying days over the interval

Table 5.1: Comparison of the lost flying days incurred for each aircraft within the study between the A-check and the matheuristic

AC	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	**Total**
A-Check	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	**15**
Mat.	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**0**

AC: Aircraft, Mat.: Matheuristic

Figure 5.4 and table 5.2 present the results on the lost flying hours incurred during the experimental study for both tested approaches. Lost flying hours are incurred when a task is executed before it is due, and therefore its task interval is not fully utilized. This will result in extra required maintenance over the aircraft's lifetime. To determine the lost flying hours for the A-check strategy, an approximation had to be made, using the average flight hours per day. This means that if an A-check was performed on day 12, it is assumed that the aircraft had traveled a total number of flight hours at that point of 12 times the average flight hours per day. The lost flying hours of any task were then calculated by calculating the difference between their remaining number of legal flight hours at day 0 and the number of traveled flight hours on the day of the planned A-check.

As seen in figure 5.4 and read from table 5.2, there is a very substantial difference between the two methods in terms of lost flying hours. The number of 219.56 thousand lost flying hours, incurred with the A-check planning, decreased to 4.32 thousand lost flying hours when implementing the matheuristic, which equals a drop of over 98%. This drop effectively translates to a much more efficient maintenance planning, where tasks are performed when they are due, instead of when the latest A-check opportunity occurs. If the task intervals are taken into account, and the non-core task intervals are generated following a uniform distribution with a lower bound of two months and an upper bound of twelve months the average interval utilization of the A-check strategy came out at roughly 91%. When using the proposed matheuristic, this average interval utilization rose to over 99%. This substantial increase in average interval utilization for the A-checks means that less maintenance will need to be performed over the aircraft's life cycle.

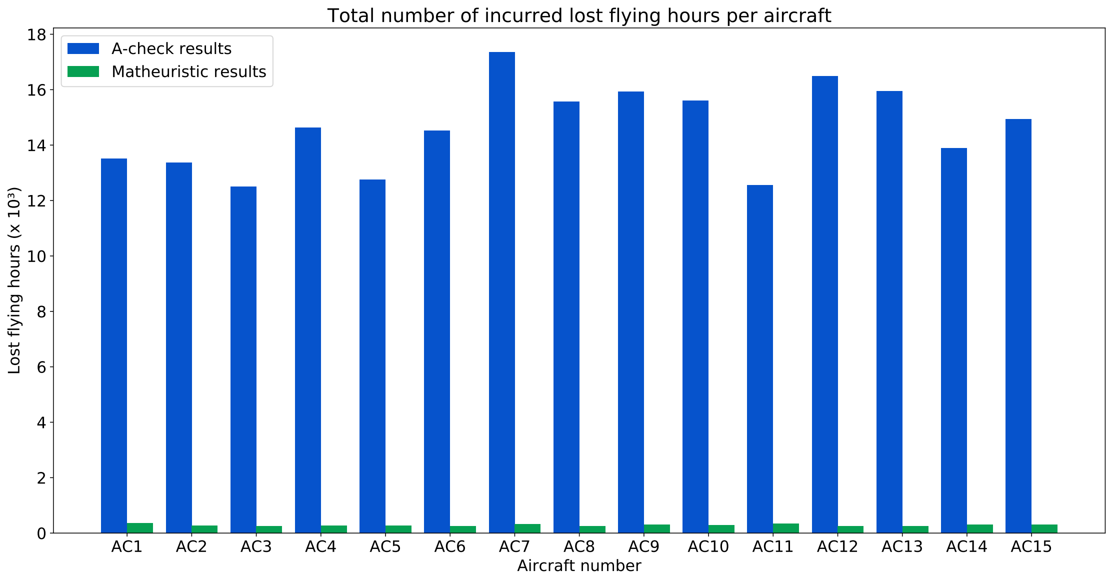


Figure 5.4: Visualization of the lost flying hours incurred per aircraft

Table 5.2: Comparison of the lost flying hours (in thousands) incurred for each aircraft within the study between the A-check and the matheuristic

AC	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	**Total**
A-Check	13.51	13.37	12.51	14.63	12.75	14.51	17.36	15.56	15.93	15.60	12.56	16.50	15.95	13.88	14.94	**219.56**
Mat.	0.36	0.28	0.25	0.27	0.27	0.26	0.33	0.25	0.30	0.29	0.34	0.26	0.26	0.31	0.30	**4.32**

AC: Aircraft, Mat.: Matheuristic

Figure 5.5 presents the results on the man-hours at the maintenance station for every night in the 60-day interval. When using the A-check strategy, every four days there is large peak of around 100 man-hours, which is the result of all the tasks in the A-check being performed on that day. In between two consecutive checks none of the considered maintenance tasks are executed. The results of the matheuristic show a more phased maintenance planning approach, because maintenance tasks can now be performed on overnight stays at the maintenance station. There are still peaks visible on the same days the previously scheduled A-checks, although they are considerably smaller. These peaks are the result of the assumption that we made at the start of this chapter, namely that half the tasks in the A-check are part of a core set of tasks that share a due date with the previously scheduled A-checks. Due to the many tasks in this core set, the matheuristic has aimed to construct a route, for each aircraft, that spends the night at the maintenance station on the same day as the A-check was previously scheduled. An advantage of a more phased maintenance approach is that the mechanics do not experience the same peaks in workloads as before.

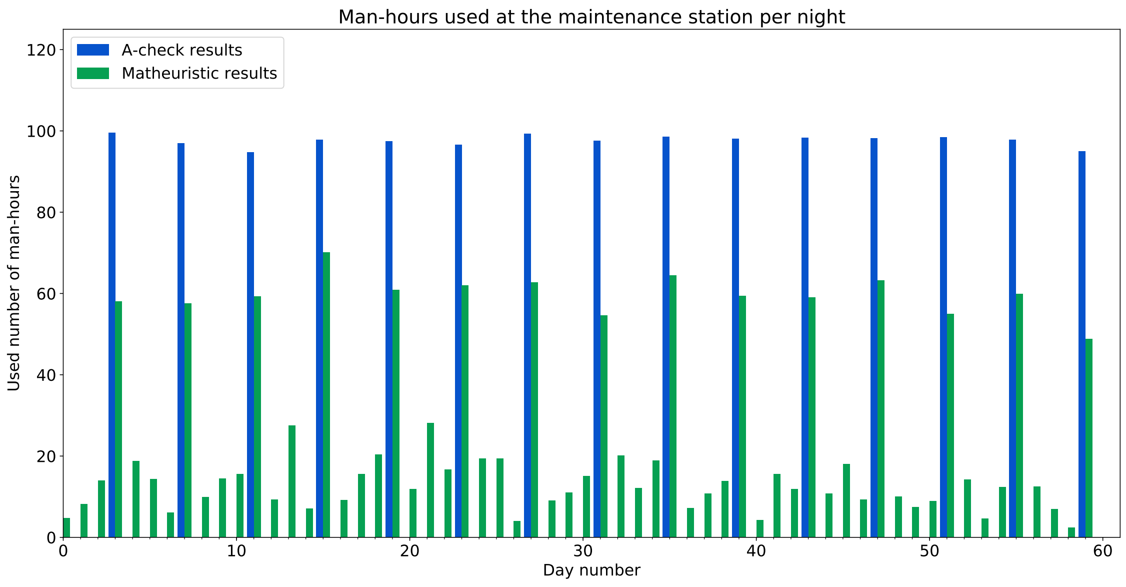


Figure 5.5: Visualization of the used man-hours at the maintenance station per day

5.2. Sensitivity Analysis

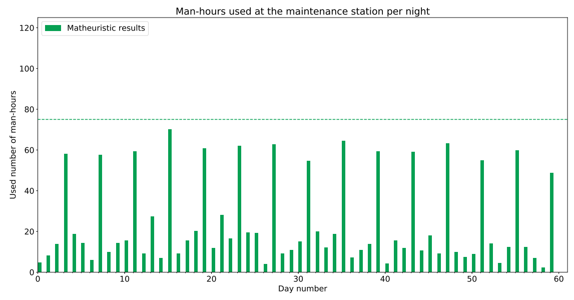
A sensitivity analysis is conducted to examine how the available man-hours per night at the maintenance station for the considered maintenance tasks affect the results of the selected matheuristic. Four different scenario's will be tested. In the first scenario, 75 man-hours are available every night, in the second scenario 50 man-hours, in the third scenario 40 man-hours and in the final scenario 35 man-hours. The results of the different scenarios in terms of lost flying hours are given in table 5.3. The corresponding used man-hours per day at the maintenance station for each scenario are presented in figure 5.6. The first thing to notice is that the matheuristic was not able to find a feasible solution on the 13th day to the routing and scheduling problem with a maximum capacity of 35 man-hours per night at the station. The availability of 35 man-hours every night would theoretically be enough to plan all the considered maintenance tasks. However, within the 7-day planning horizon around day 13 there were not enough night layovers at the maintenance station to construct a feasible schedule.

Table 5.3: Lost flying hours incurred with changing station capacity parameter (in man-hours)

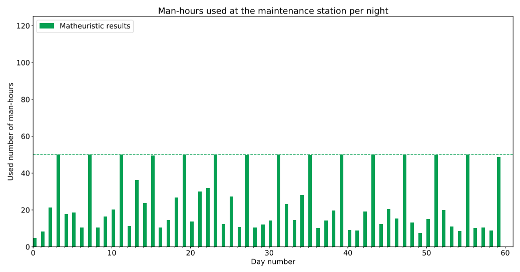
Station Capacity	Lost Flying hours
75	4316
50	6178
40	7093
35	-

Table 5.3 indicates a clear trend that increasing the station capacity has a positive effect on the incurred number of lost flying hours. In figure 5.6a it is seen that on many occasions the matheuristic plans maintenance nights close to 75 man-hours. When decreasing the capacity to 50 and even 40 man-hours, see figure 5.6b and 5.6c, a number of the tasks will have to be moved forward, resulting in a larger number of lost flying hours. Decreasing the station capacity leads to a more phased maintenance

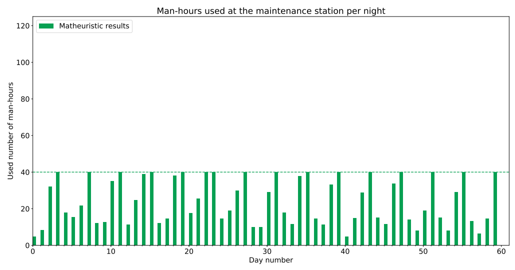
planning, because the existence of peaks above a certain threshold is forbidden. Decreasing the station capacity past a specific point will lead to infeasibility in the model however. Where this feasibilty point lies, depends on the number of tasks to be planned and on the number of available night stays at the maintenance station in the flight schedule.



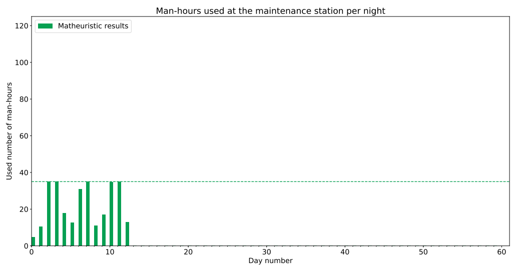
(a) Used man-hours per night with a station capacity of 75 man-hours per night



(b) Used man-hours per night with a station capacity of 50 man-hours per night



(c) Used man-hours per night with a station capacity of 40 man-hours per night



(d) Used man-hours per night with a station capacity of 35 man-hours per night

Figure 5.6: Visualization of the used man-hours per night at the maintenance station for varying station capacities

5.3. Discussion of the Experimental Results

In this experimental study, the planning results of a standard A-check strategy are compared to the results of our proposed individual task-based model for a fleet of 15 aircraft. The rolling horizon matheuristic with one day exact forecasting was selected as the solution method, because it proved to generate good results within reasonable times for similar fleet sizes. We analyzed the two methods over an interval of 60 days. The first result that stands out is the difference in lost flying days. When using our proposed matheuristic, it is no longer necessary to take an aircraft out of operations for one day once every ±60 days. This increase in aircraft availability can be used in multiple ways and it is up to the airline to decide which way suits their needs the best. One straightforward way is to use this extra availability to expand the airline's flight schedule and operate more flights. Depending on the aircraft's utilization level, a day of operations may represent between \$75k and \$120k of additional revenue [11]. If the assumption is made that an aircraft requires 6 A-checks per year, the extra revenue could equal up to \$ 7.2 million per aircraft over a time period of 10 years. Further, this only considers the decreased downtime due to A-checks. It is possible that number of lost flying days due to C-checks, and perhaps even D-checks, could be decreased as well, although it is unlikely it will fall to zero.

In the study, the number of lost flying hours decreased with over 98%. This drop is due to the different strategy that an individual task-based maintenance strategy follows, namely that tasks are performed when they are due, instead of when the latest A-check opportunity occurs. It is more difficult to put a price tag on the saved costs due to the decreased number of lost flying hours. It can, however, logically be deduced that better utilizing the maintenance tasks' intervals leads to less required maintenance in the long run. This could express itself in a smaller mechanics team being able to handle the same fleet, and could provide savings in salary costs.

An A-check strategy results in distinct peaks of used man-hours at the maintenance station. For some time, none of the considered maintenance tasks are performed, and at the given time they are all performed together. The results of our presented matheuristic present a more phased maintenance planning. The removal of distinct maintenance peaks could lead a more satisfactory working environment. The adoption of an individual task-based planning strategy could complicate the construction of schedules for the mechanics however. Furthermore, from a conducted sensitivity analysis, it could be seen that decreasing the maintenance station's capacity is accompanied by a more phased maintenance schedule, but also more lost flying hours.

6 Impact on the Aircraft Maintenance Industry

Airlines need to find ways to improve their operational efficiency due to intensified competition among airlines in the previous decades. Even though maintenance, repair, and overhaul (MRO) represent a significant portion of an airline's operational costs (at least 9% in 2018 excluding overhead costs [18]), the planning process is far from optimized. In the industry, the planning of aircraft maintenance is often still a very manual process, relying on the experience of the planners.

The scheduling of aircraft maintenance is not a stand-alone problem. Because maintenance can only be done at certain prepared airports (referred to as maintenance stations), the scheduling of maintenance is codependent with the aircraft's routing. This and other codependencies cause the scheduling of an airline's operations and resources to be a very complex problem, which is why it is common practice to break up the resource planning into several sequential stages. The five classes in which the airline industry's different planning and scheduling problems are divided are flight scheduling, fleet assignment, aircraft maintenance routing, crew scheduling, and tail assignment [24]. In the flight scheduling stage, an airline decides which flights it will operate, and during the fleet assignment, it is determined what type of aircraft will operate each flight. The flight schedule is created approximately one year in advance [21]. During the aircraft maintenance routing (AMR), individual flights are combined to form routes. It has been given this name because planners try to create flight routes that are feasible for smaller and more frequently occurring maintenance tasks [27]. During the crew scheduling stage, crew members are assigned to these routes with the aim to minimize crew cost and maximize various other objectives, such as quality of life and crew satisfaction. The crew schedules are determined approximately one month before the day of operations [21]. Finally, the tail assignment (TA) is solved, where the created routes are assigned to specific aircraft (often referred to as tail numbers). During the tail assignment, an aircraft's initial location and its individual maintenance needs are the key issues, since the route that is assigned in this stage needs to be compliant with the aircraft's individual maintenance needs. Adjustments to the routes created in the AMR might be required because these considerations and possible schedule disruptions could result in an infeasible maintenance schedule.

Even though, since the coming of the Boeing 777, it is no longer mandated, most airlines currently still plan the better part of their MRO activities in extensive checks, called letter-checks (A, B, C, or D). A typical A-check includes the inspection of the interior and exterior of the aircraft. C-checks include the inspection and functionality-testing of individual systems and components. During a D-check, the structurally important components are inspected, which requires the uncovering of the airframe, wings, and supporting structure [3]. The interval with which these letter-checks must be performed ranges from roughly two months (A-check) up to six years (D-check) [20], although these intervals may differ somewhat between aircraft and airlines. During a letter-check, the aircraft is taken out of operations, and within this period, an extensive number of maintenance tasks are executed at once. Letter-checks are planned within specific pre-made slots. Throughout the year there are many A-check slots, spanning an entire day, that can be reserved for specific aircraft. If it is predicted that an aircraft will be due for an A-check, an A-check slot can be reserved for it. Usually, A-check slots are reserved about one month in advance. Slots for C- and D-checks are fewer and claimed much further in advance because they require substantially more downtime.

The planning of aircraft maintenance is a complicated process, and the strategy of using letter-checks is a proven method to simplify the process. However, the use of letter-checks for maintenance planning has two significant downsides: (1) the aircraft must be taken out of operations, and (2) many maintenance tasks are performed considerable time before they are due. The first downside leads to a decrease in aircraft availability and the second downside to more required maintenance operations during the aircraft's lifetime.

This thesis explores the design, feasibility, and impact of a different maintenance planning strategy, namely one in which tasks are planned individually within available ground times in the flight schedule, rather than be grouped inside letter-checks for which the aircraft must be taken out of operations. In this work, a model was created in which the tail assignment is solved, and the individual tasks are planned during overnight stays on the maintenance station. This is done by partitioning lines (a sequence of day-routes that start and/or end at the maintenance station) to each individual aircraft that allow them to have opportunities for maintenance if they require it. Each maintenance task is given a due date and a number of legal remaining flight hours. The objective of the model is to allocate the lines to the aircraft in such a way that tasks can be planned as *late* as possible. By planning tasks as late as possible, the task intervals will be fully utilized and the required maintenance over the aircraft's lifetime, and the costs that come with it are reduced.

In this work, multiple solution approaches were designed and tested to solve our presented model. One of the designed solution approaches that produced excellent results is a rolling horizon matheuristic with exact forecasting. In an experimental study, the maintenance scheduling results of a standard A-check planning and the results of the rolling horizon matheuristic with one day exact forecasting were compared over an interval of 60 days for a fleet of 15 aircraft. Where the A-check planning required every aircraft to be taken out of operations for one day to complete the check, the matheuristic was able to plan all required maintenance on overnight stays at the maintenance station. As a result, the aircraft did not have to be taken out of operations. This increase in aircraft availability can be used in multiple ways, and it is up to the airline to decide which way suits their needs the best. One straightforward way is to use this increased availability to expand the airline's flight schedule and operate more flights. Depending on the aircraft's utilization level, a day of operations may represent between \$75k and \$120k of additional revenue [11]. If the assumption is made that an aircraft currently requires 6 A-checks per year, the extra revenue could equal up to \$ 7.2 million over a time period of 10 years per aircraft. Further, this amount only reflects the potential savings on A-checks. Using an individual task-based strategy could possibly reduce the required downtime for C-checks or possibly even D-checks as well. If an airline is not looking to expand its flight schedule, the increased availability could possibly allow the airline to operate its same schedule with fewer aircraft, in which case one or more aircraft could be sold. The feasibility of this depends on the fleet size and the design of the airlines' current flight schedule. A third way in which an airline could use this increased aircraft availability to its benefit is by increasing its reserve fleet. By stationing a reserve aircraft and a reserve crew, an airline can decrease the effect of potential delays and increase customer satisfaction.

Secondly, in the experimental study, the number of lost flying hours were compared between the standard A-check planning and our matheuristic. A lost flying hour is defined as a flying hour between the execution of the task and when the task is actually due. Lost flying hours are incurred when maintenance tasks are performed before they are due, so if a task is due after 500 flight hours, and it is performed after 450 flight hours, 50 lost flying hours have been incurred. Repeatedly incurring lost flying hours, because maintenance tasks are planned before they are due, will lead to more required maintenance operations over the aircraft's lifetime. In the experimental study, the number of *lost flying hours* decreased with over 98%, when using our proposed matheuristic, as opposed to the standard A-check planning. In our experimental study, this corresponded to an increase in the task interval utilization of over 8%. It is more difficult to put a price tag on the saved costs due to the more efficient maintenance planning. However, as stated before, the better utilization of the maintenance tasks' intervals leads to less required maintenance in the long run. This could express itself in a smaller mechanics team that could handle the same fleet and could provide savings in salary costs.

The experimental study also showed that a standard A-check planning is accompanied by distinct peaks in workload. Using an individual task-based maintenance planning results in a more phased maintenance approach which attenuates the peaks and creates a more balanced workload distribution.

A final aspect in which implementing our proposed model and solution method could impact the airline industry is the role of the maintenance planner. As stated at the start of this chapter, the planning of aircraft maintenance is often still a very manual process, relying on the experience of the planners. The combination of the planners' required experience and their importance to the day-to-day operations makes it difficult to recruit new employees for this job and has, in some cases, lead to an aging staff in this department. Implementing a system that can automate part of their job thus not only has the potential to save in salary costs but might actually prove instrumental to an airline's continuous operations in the long term.

Certain changes will have to be made within an airline's current operations to implement our model successfully. One of these required changes is increased flexibility in the roster scheduling for the mechanic workforce. As shown in the experimental study, the required man-hours at the maintenance station for a standard A-check strategy can be planned far in advance because the A-check slots are determined long before the day of operations. When implementing our model, the required man-hours at the maintenance station are much more variable and are only determined when solving the tail assignment. The tail assignment is solved much closer to the day of operations, which means that the mechanic workforce's schedules will be fixed closer to the day of operations as well.

A second condition that has to be met in order to implement our model successfully is to create flight schedules in which there are plenty of night stays at the maintenance station. When the objective is to plan maintenance as close to the due date as possible, it is required that aircraft can be assigned routes that let them spend the night at the maintenance station whenever they need it. Most airlines operating with a hub-and-spoke system are likely to satisfy this condition since every flight cycle starts and ends at the hub. However, when the flight schedules force aircraft to spend the night at a different station than the hub very often, the model might experience difficulties finding maintenance-feasible routes for each aircraft in the fleet.

7 Conclusion & Recommendations

This chapter presents the conclusions of this research and recommendations for further research. First, the research questions are answered in section 7.1. Secondly, in section 7.2, recommendations for future research are given.

7.1. Conclusion

This research was conducted with the objective of enriching the academic community by developing a model with the capability of optimizing the maintenance and corresponding flight schedules for aircraft using an individual task-based approach. To comply with the sequential nature of an airline's scheduling operations, the scope was limited to the final scheduling stage, namely the tail assignment. This means that this work did not look at the design of the flight schedule but was restricted to the assignment of aircraft to pre-constructed routes. The main research question was formulated as:

How can the efficiency and quality of the maintenance scheduling process in the airline industry be improved through optimization?

From the conducted literature review in this thesis, we conclude that airlines typically use a letter-check-based maintenance scheduling strategy. We also conclude that the majority of academic research has focused on the scheduling optimization of generic maintenance checks that must be performed at least once every *d* days. The few works that did consider a wider range of maintenance requirements proposed solutions that would conflict with airlines' overall scheduling framework. After careful consideration, we decided that the practice of assigning day-routes to aircraft was the best modeling approach for our application. A total of four possible mathematical formulations used in the literature for the routing and maintenance scheduling of aircraft were identified and examined. We conclude that, out of these four formulations, the city-day network would provide the most compact model while meeting the requirements.

To design a working model that could successfully optimize an individual task-based maintenance schedule, numerous considerations had to be taken into account. Firstly, to warrant the feasibility of the schedule, an aircraft needs to spend the night at a maintenance station at least once every *d* days. For this reason, the presented model assigns *lines* (a sequence of day-routes that start and/or end at a maintenance station) instead of single day-routes. These lines are created during the AMR, an earlier stage in the overall scheduling framework, and are taken as input for our model. Because the planners take the given constraint into account during the creation of these lines, no lines are created that go longer than *d* days without spending the night at a maintenance station. A second consideration that is taken into account is the fact that the interval of a task can be given both in days as well as in flight hours. This expresses itself in the model by the inclusion of a due date and a number of remaining legal flight hours for each task. A third consideration is the capacity of the maintenance station. in the model, each task was given a size expressed in the number of man-hours it requires to complete. The capacity constraint assigns a finite number of man-hours for the maintenance station every night. Because it is difficult to precisely define a cost for maintenance, the surrogate objective of minimizing the lost flying hours was used. This surrogate objective stems from the reasoning that performing a task before it is due will lead to more required maintenance and, thus, more costs in the long run.

Because an exact solution method proved unsuited for larger problem sizes, we designed several matheuristics to solve the given problem. The created matheuristics are divided into two categories: decomposition by aircraft matheuristics and rolling horizon matheuristics. Within the former category, two matheuristics were created: a depth-first and random search matheuristic, inspired by the work of Sriram and Haghani [40]; and a successive route checking matheuristic. Within the latter category, three different matheuristics were designed: a rolling horizon matheuristic without forecasting, a rolling horizon matheuristic with exact forecasting and a rolling horizon matheuristic with average flight hour forecasting. All matheuristics were first tested for smaller problem sizes and their results compared to the exact method's results. The results showed that all of the rolling horizon matheuristics consistently outperformed their decomposition by aircraft counterparts and that they generated good solutions in quick fashion. We therefore conclude that rolling horizon matheuristics are the better fit to our problem.

Out of all the rolling horizon matheuristics, especially the ones with exact forecasting produced excellent results. The results improved when expanding the forecasting horizon. However, expanding the exact forecasting horizon had a significant impact on the required computation times; therefore, results were only presented of exact forecasting horizons up to two days. The rolling horizon with average flight hours forecasting matheuristic, on average, offered better results than the rolling horizon matheuristics without forecasting. One notable aspect of this, however, was that the results did not consistently improve when expanding the forecasting period.

All matheuristics were also tested for larger problem sizes of up to 25 aircraft and 20 possible tasks for each aircraft. There were no exact solutions available for these larger problem sets, but the results could still be compared between the different solution approaches. Again, both decomposition-by-aircraft methods performed poorly in comparison to the rolling horizon matheuristics. The rolling horizon methods that produced the solutions with the best objective values were again the matheuristics that used exact forecasting. For the largest tested problem sets, however, the exact forecasting method became very time consuming, and no solutions within reasonable time were found for a forecasting period of two days when using problem sets containing 20 or more aircraft. The rolling horizon matheuristics with average flight hour forecasting performed, on average, slightly better than the rolling horizon matheuristic without forecasting. Similarly to the smaller problem sets, the results did not show an improving trend when expanding the average flight hour forecasting horizon. We conclude that this is most likely due to a trade-off between the positive effects of getting information on a larger part of the model and the negative effects of decreasing prediction accuracy. The answer to the question of which rolling horizon matheuristic is best suited to solve the presented individual task-based maintenance planning depends on several factors: (1) the size of the fleet that an airline operates; (2) the number of tasks that they wish to plan on an individual basis; and (3) the available computing power. If computationally feasible for the airline, the rolling horizon matheuristics with exact forecasting provide the best results. If the problem set is of such size that exact forecasting is not feasible, it is concluded that the rolling horizon matheuristic with average flight hour forecasting is the most suitable choice for larger fleets.

An experimental study was performed in which the results of a standard A-check planning and the results of the rolling horizon matheuristic with one-day exact forecasting were compared with one another. The results showed that for a fleet of 15 aircraft and a time period of 60 days, the total number of downtime days for all aircraft could be reduced from 15 days to 0 by switching from the A-check strategy to our proposed matheuristic. Furthermore, the total number of weighted lost flying hours was reduced by over 98%, meaning that tasks were planned much closer to their due dates. This led to a substantial increase in the average task interval utilization. Finally, the distinct workload peaks that accompanied the A-check strategy were attenuated, resulting in a more phased and balanced workload distribution.

Finally, the potential impact of our proposed model and solution method on the airline industry was presented. Depending on the aircraft's utilization level, a day of operations may represent between \$75k and \$120k of additional revenue [11]. Under the assumption that an aircraft requires six A-checks per year, removing the need to take aircraft out of operations for an A-check could improve revenues for a single aircraft up to $7.2 million over a period of 10 years. Further, this amount only reflects the potential savings on A-checks. Using an individual task-based strategy could possibly reduce the required downtime for C-checks and possibly even D-checks as well. The decrease in lost flying hours leads to better utilization of the maintenance tasks' intervals, which results in less required maintenance in the long run. This could express itself in a smaller mechanics team being able to handle the same fleet and could provide savings in salary costs. A third area in which our work could impact the industry is the role of the maintenance planner. Currently, this is often still a very manual process that requires much experience of the planners. Automating parts of the planning process can lead to reduced salary costs and reduce the required experience-barrier that new planners face.

Certain changes will have to be made within the current operations of an airline to implement our model successfully. It is concluded that the mechanic workforce scheduling must become more flexible so that their schedules are dependent on the need for maintenance instead of the other way around. It is also concluded that to implement our model successfully, there must be enough maintenance opportunities within the flight schedules, so that aircraft that require maintenance can be assigned a route that lets them spend the night at the maintenance station when needed.

7.2. Recommendations for Further Research

There has been relatively little research performed on optimizing the aircraft maintenance scheduling and corresponding routing on an individual task basis. Therefore several possible future research directions could be further explored.

The first recommendation for further research is to add an overtime component to the maintenance station capacity. In some cases, it might be beneficial, or even necessary, to exceed the given man-hours of the station capacity. This could practically be done by letting mechanics work overtime or by deploying a larger mechanics team for that night. Planning overtime could be represented in the objective function by a penalty. Apart from gaining more flexibility in maintenance scheduling, an airline could even use this extension to get a better view of the required station capacity.

The second recommendation for further research is to dive deeper into the nature of the different maintenance tasks. In practice, some maintenance tasks could have good synergy with each other, and therefore it might be beneficial to execute them together. This could be the case if the components related to the maintenance tasks physically lie close to each other in the aircraft. Also, in practice, some maintenance tasks often require a different type of mechanic than other tasks. The mechanic requirement could be given as a task property in the model to make sure that there are always enough engineers of every required type available.

Finally, the third recommendation for further research could be to include ground times during the day for maintenance planning as well. This would allow for more potential maintenance opportunities to be utilized. However, also considering ground times during the day as maintenance opportunities would increase the size and complexity of the model tremendously.

Bibliography

[1] URL https://www.schiphol.nl/en/schiphol-as-a-neighbour/page/flight-paths-and-runway-use/. Visited 28-4-2020.

[2] URL https://www.heathrow.com/company/local-community/noise/operations/night-flights. Visited 28-4-2020.

[3] S. P. Ackert. Basics of Aircraft Maintenance Programs for Financiers. Technical report, 2010. URL http://aircraftmonitor.com/uploads/1/5/9/9/15993320/basics_of_aircraft_maintenance_programs_for_financiers___v1.pdf. Accessed 20-10-2019.

[4] Agostinho Agra, Marielle Christiansen, Alexandrino Delgado, and Luidi Simonetti. Hybrid heuristics for a short sea inventory routing problem. *European Journal of Operational Research*, 236: 924–935, 08 2014. doi: 10.1016/j.ejor.2013.06.042.

[5] A. Ahmadi, P. Söderholm, and U. Kumar. An overview of trends in aircraft maintenance program development: Past, present, and future. In *Proceedings of the European Safety and Reliability Conference 2007, ESREL 2007 - Risk, Reliability and Societal Safety*, volume 3, pages 2067–2075, 2007.

[6] C. Archetti and M. G. Speranza. A survey on matheuristics for routing problems. *EURO Journal on Computational Optimization*, 2(4):223–246, 2014.

[7] Cynthia Barnhart, Natashia Boland, Lloyd Clarke, Ellis Johnson, George Nemhauser, and Rajesh Shenoi. Flight string models for aircraft fleeting and routing. *Transportation Science*, 32:208–220, 08 1998. doi: 10.1287/trsc.32.3.208.

[8] Mehmet Basdere and Umit Bilge. Operational aircraft maintenance routing problem with remaining time consideration. *European Journal of Operational Research*, 235:315–328, 05 2014. doi: 10.1016/j.ejor.2013.10.066.

[9] Eduardo Camacho and Carlos Bordons. *Model Predictive Control*, volume 13. 01 2004. ISBN 978-0-85729-398-5. doi: 10.1007/978-0-85729-398-5.

[10] Amy Cohn and Marcial Lapp. *Airline Resource Scheduling*. 01 2011. ISBN 9780470400531. doi: 10.1002/9780470400531.eorms0020.

[11] Q. Deng, B. F. Santos, and R. Curran. A practical dynamic programming based methodology for aircraft maintenance check scheduling optimization. *European Journal of Operational Research*, 281(2):256–273, 2019.

[12] Duarte Dinis, Ana Barbosa-Povoa, and A.P. Teixeira. A supporting framework for maintenance capacity planning and scheduling: Development and application in the aircraft mro industry. *International Journal of Production Economics*, 218:1–15, 05 2019. doi: 10.1016/j.ijpe.2019.04.029.

[13] A. Erdmann, A. Nolte, A. Noltemeier, and Rainer Schrader. Modeling and Solving an Airline Schedule Generation Problem. *Annals of Operations Research*, 107:117–142, 10 2001. doi: 10.1023/A:1014998931654.

[14] Thomas Feo and Jonathan Bard. Flight scheduling and maintenance base planning. *Management Science*, 35:1415–1432, 12 1989. doi: 10.1287/mnsc.35.12.1415.

[15] C.W. Gits. Design of maintenance concepts. *International Journal of Production Economics*, 24(3): 217 – 226, 1992. ISSN 0925-5273. doi: https://doi.org/10.1016/0925-5273(92)90133-R.

[16] R. Gopalan and K. T. Talluri. The aircraft maintenance routing problem. *Operations research*, 46 (2):260–271, 1998.

[17] C. A. Hane, C. Barnhart, E. L. Johnson, R. E. Marsten, G. L. Nemhauser, and G. Sigismondi. The fleet assignment problem: Solving a large-scale integer program. *Mathematical Programming*, 70 (1-3):211–232, 1995.

[18] IATA's Maintenance Cost Technical Group. Airline maintenance cost executive commentary. Technical report, International Air Transport Association, 2019.

[19] Oumaima Khaled, Michel Minoux, Vincent Mousseau, Stéphane Michel, and Xavier Ceugniet. A compact optimization model for the tail assignment problem. *European Journal of Operational Research*, 264(2):548–557, 06 2017.

[20] A.H Kinnison and S. Siddiqui. *Aviation Maintenance Management*. McGraw Hill, 2012.

[21] Diego Klabjan. *Large-Scale Models in the Airline Industry*, pages 163–196. 01 2005. doi: 10.1007/0-387-25486-2_6.

[22] Eve Lacasse-Guay, Guy Desaulniers, and François Soumis. Aircraft routing under different business processes. *Journal of Air Transport Management*, 16:258–263, 09 2010. doi: 10.1016/j.jairtraman.2010.02.001.

[23] Marcial Lapp and Florian Wikenhauser. Incorporating aircraft efficiency measures into the tail assignment problem. *Journal of Air Transport Management*, 19:25 – 30, 2012. doi: https://doi.org/10.1016/j.jairtraman.2011.12.006.

[24] Zhe Liang and Wanpracha Chaovalitwongse. *Optimization and Logistics Challenges in the Enterprise*, volume 30, chapter The Aircraft Maintenance Routing Problem, pages 327–348. Springer, Boston, MA, 05 2009.

[25] Zhe Liang and Wanpracha Chaovalitwongse. A network-based model for the integrated weekly aircraft maintenance routing and fleet assignment problem. *Transportation Science*, 47:493–507, 11 2013. doi: 10.1287/trsc.1120.0434.

[26] Zhe Liang, Wanpracha Chaovalitwongse, Huei Huang, and Ellis Johnson. On a new rotation tour network model for aircraft maintenance routing problem. *Transportation Science*, 45:109–120, 02 2011. doi: 10.1287/trsc.1100.0338.

[27] Zhe Liang, Yuan Feng, Xiaoning Zhang, Tao Wu, and Wanpracha Chaovalitwongse. Robust weekly aircraft maintenance routing problem and the extension to the tail assignment problem. *Transportation Research Part B: Methodological*, 78:238–259, 08 2015. doi: 10.1016/j.trb.2015.03.013.

[28] Rogier Lieshout, Paolo Malighetti, Renato Redondi, and Guillaume Burghouwt. The competitive landscape of air transport in europe. *Journal of Transport Geography*, 50:68–82, 06 2015. doi: 10.1016/j.jtrangeo.2015.06.001.

[29] Rongfang Liu and Nadereh Moini. Benchmarking transportation safety performance via shift-share approaches. *Journal of Transportation Safety Security*, 7:124–137, 04 2015.

[30] S. J. Maher, G. Desaulniers, and F. Soumis. Recoverable robust single day aircraft maintenance routing problem. *Computers and Operations Research*, 51:130–145, 2014.

[31] S. J. Maher, G. Desaulniers, and F. Soumis. The daily tail assignment problem under operational uncertainty using look-ahead maintenance constraints. *European Journal of Operational Research*, 264(2):534–547, 2018.

[32] Walid Moudani. A dynamic approach for aircraft assignment and maintenance scheduling by airlines. *Journal of Air Transport Management*, 6:233–237, 10 2000. doi: 10.1016/S0969-6997(00)00011-9.

[33] J. Quehl, U. Müller, and F. Mendolia. Short-term annoyance from nocturnal aircraft noise exposure: results of the norah and strain sleep studies. *International archives of occupational and environmental health*, 90(8):765–778, 2017.

[34] Jørgen Rakke, Magnus Stålhane, Christian Moe, Marielle Christiansen, Henrik Andersson, Kjetil Fagerholt, and Inge Norstad. A rolling horizon heuristic for creating a liquefied natural gas annual delivery program. *Transportation Research Part C Emerging Technologies*, 19:896–911, 08 2011. doi: 10.1016/j.trc.2010.09.006.

[35] Sebastian Ruther, Natashia Boland, Faramroze Engineer, and Ian Evans. Integrated aircraft routing, crew pairing, and tail assignment: Branch-and-price with many pricing problems. *Transportation Science*, 51:177–195, 08 2016. doi: 10.1287/trsc.2015.0664.

[36] Nima Safaei and Andrew Jardine. Aircraft routing with generalized maintenance constraints. *Omega*, 80:111–122, 09 2017. doi: 10.1016/j.omega.2017.08.013.

[37] A. Sarac, R. Batta, and C. M. Rump. A branch-and-price approach for operational aircraft maintenance routing. *European Journal of Operational Research*, 175(3):1850–1869, 2006.

[38] C. Senturk and I. Ozkol. The effects of the use of single task-oriented maintenance concept and more accurate letter check alternatives on the reduction of scheduled maintenance downtime of aircraft. *International Journal of Mechanical Engineering and Robotics Research*, 7(2):189–196, 2018.

[39] Caner Senturk, Mehmet Serif Kavsaoglu, and Melike Nikbay. Optimization of aircraft utilization by reducing scheduled maintenance downtime. In *10th AIAA Aviation Technology, Integration, and Operations (ATIO) Conference*.

[40] Chellappan Sriram and Ali Haghani. An optimization model for aircraft maintenance scheduling and re-assignment. *Transportation Research Part A: Policy and Practice*, 37:29–48, 01 2003. doi: 10.1016/S0965-8564(02)00004-6.

[41] Kenneth Sörensen. Metaheuristics – the metaphor exposed. *International Transactions of Operations Research*, In Press:3–18, 01 2013. doi: 10.1111/itor.12001.

[42] Xin Tian, R.R. Negenborn, Peter-Jules Overloop, J.M. Maestre, Anna Sadowska, and Nick van de Giesen. Efficient multi-scenario model predictive control for water resources management with ensemble streamflow forecasts. *Advances in Water Resources*, 109:58–68, 09 2017.

[43] M. van der Windt. Curfews at international airports - a study of current practice. In *28th Congress of the International Council of the Aeronautical Sciences 2012, ICAS 2012*, volume 6, pages 5115–5116, 2012.

A

Scientific Paper

Matheuristics for the dynamic optimization of the routing and maintenance scheduling of aircraft for individual tasks

A.S.A. Steenkamp, F. Schulte and R.R. Negenborn

Department of Transportation Engineering & Logistics – Delft University of Technology, Mekelweg 2, 2628 CD Delft, Netherlands

ARTICLE INFO

Keywords:
Aircraft routing
Maintenance scheduling
Matheuristics
OR in airlines

ABSTRACT

Increasing competition amongst airlines necessitates them to improve the efficiency of their operations. Even though maintenance, repair, and overhaul (MRO) represent a significant portion of an airline's operational costs, the scheduling of these operations is often still suboptimal. Usually, many maintenance tasks are congregated in large checks, causing a need for the aircraft to be taken out of operations and tasks to be executed before they are due. The purpose of this study is to develop a methodology that provides flight routes to aircraft and individually plans the maintenance tasks on nocturnal overlays within these routes over a given planning horizon with the objective of maximizing the utilization of the total remaining flying time of the fleet. For this purpose, we develop a mixed integer programming (MIP) model based on a city-day network representation. Multiple matheuristics have been developed to provide good solutions in reasonable computation times. During an experimental study one of the selected matheuristics was able to remove the need for aircraft to be taken out of operations, and decrease the lost flying time, incurred by planning tasks before they are due, by over 98%. Our presented approach can be used by mid-sized airlines to optimize their maintenance schedules by increasing aircraft availability and reducing maintenance costs.

1. Introduction

Since the deregulation of the intra-European air transport market in 1997, competition among airlines has intensified significantly [18]. From that moment onwards, any EU air carrier was allowed to operate from any EU country. Low-cost carriers, improvement of high-speed rail networks, and increased price transparency provided by the internet have all contributed to this increase of competition. These days, passengers have a broader choice in terms of routing and pay a lower price. Airlines need to find ways to improve their operational efficiency in the current competitive market of air transport. Even though maintenance, repair, and overhaul (MRO) represent a significant portion of an airline's operational costs (at least 9% in 2018 excluding overhead costs [11]), the planning process is far from optimized. In the industry, it is typically still a manual process, producing suboptimal solutions.

For each aircraft, there is a distinguishable list of maintenance tasks that are to be carried out before a specific due date, with a repetitive interval. Each task has a unique task code, a determined amount of necessary man-hours for completion, an account of the required types of mechanics, and much more practical information. If a particular maintenance task has not yet been performed after its due date has passed or after its remaining legal flight hours have run out, the aircraft loses its airworthiness, rendering it unsuitable for safe flight until the task is performed.

Currently, most airlines plan the bulk of their MRO activities in extensive checks, called letter-checks (A, B, C, or D). During a letter-check, the aircraft is taken out of operations, and many maintenance tasks are executed together. In between the intervals of letter-checks, an aircraft still needs occasional maintenance checks every couple of days for more frequently occurring tasks. The main advantage of using extensive letter-checks lies in the relative ease of the planning, as one only has to plan one event for a large number of tasks. However, the use of letter-checks for maintenance planning has two significant downsides:

1. During a check, an aircraft is taken out of operations for one or more days, resulting in a decrease in aircraft availability and a loss of potential revenue.
2. By grouping maintenance tasks with different intervals together in an extensive check, many tasks are executed before they are due, resulting in more required maintenance over an aircraft's lifetime.

These severe downsides of letter-checks have given rise to a alternatively proposed maintenance philosophy, where extensive letter-checks are dissected, and the tasks are planned at an individual level [25]. In this philosophy, every time an aircraft spends on the ground at a maintenance station is seen as an opportunity for maintenance. Maintenance planning and aircraft routing are interrelated problems. Because the execution of maintenance tasks is only possible at a maintenance station, the route that an aircraft is assigned must contain enough sizeable layovers at this station so that the individual maintenance needs for each aircraft can be met.

Because the scheduling of aircraft maintenance tasks and the routing of aircraft is an interrelated problem, it is important to consider the other scheduling problems an airline is faced with. These problems can be divided into five major classes: flight scheduling, fleet assignment, aircraft maintenance routing, crew scheduling, and tail assignment [14]. These problems are typically solved in a sequential manner, where the solution of the first problem is taken as an input for the second problem. In the flight scheduling stage, an airline decides which flights it will operate, and during the fleet assignment, it is determined what type of aircraft will operate each flight. The flight schedule is created approximately one year in advance [13]. During the aircraft maintenance routing (AMR), individual flights are combined to form routes. It has been given this name because planners try to create flight routes that are feasible for smaller and more frequently occurring maintenance tasks [17]. During the crew scheduling stage, crew members are assigned to these routes with the aim to minimize crew cost and maximize various other objectives, such as quality of life and crew satisfaction. The crew schedules are determined approximately one month before the day of operations [13]. Only a few weeks or even days (this varies strongly between airlines) before the day of operations, the tail assignment (TA) is solved, where the created routes are assigned to specific aircraft (often referred to as tail numbers). During the tail assignment, an aircraft's initial location and individual maintenance needs are the key issues, since the route that is assigned in this stage needs to be compliant with the aircraft's individual maintenance needs. Adjustments to the routes created in the AMR might be required because these considerations and possible schedule disruptions could result in an infeasible maintenance schedule. Maintenance activities for each aircraft are scheduled during ground times within their assigned routes.

Most research on aircraft maintenance scheduling and routing focuses on solving the AMR, under the assumption that each aircraft must visit a maintenance station at least once every d days for a generic check. A number of papers have attempted an integration between the AMR and the TA [20, 24, 4, 19, 17, 22, 12, 23], but in doing so they often ignore the need for crew personnel to know their work schedules ample time in advance. Furthermore, there is only scant literature that considers a wider range of maintenance requirements than the scheduling of generic daily checks [20, 24, 19, 22, 23].

In this study, our aim is to develop an individual task-based scheduling model that fits inside an airline's overall planning framework. This is done by solving the tail assignment and by planning the individual maintenance tasks within the available ground times for each aircraft, whilst aiming to minimize costs. By planning maintenance tasks at an individual level, many tasks that were previously grouped in letter checks can now be planned during ground times within their assigned flight schedule. For this purpose a new MIP formulation is proposed for the given problem. Problem sets of over ten aircraft resulted in a substantial computational burden for exact methods. Therefore, several matheuristics are developed and subsequently compared with each other. Furthermore, an experimental study is conducted to compare the results of a letter-check based scheduling approach and an individual task based scheduling approach over an interval of 60 days. Lastly, the potential impact of our model on the aircraft industry is discussed.

The paper is organized as follows: In Section 2 we present a literature review on optimization problems relating to aircraft maintenance scheduling. In Section 3, our model formulation is presented and model characteristics are discussed. In Section 4 several matheuristics are proposed and compared. In Section 5 the experimental study is presented and Section 6 discussed the potential impact on the aircraft maintenance industry.

2. Literature review

In this section we briefly discuss some of the published work on aircraft maintenance optimization problems. Table 1 presents an analysis of research published on maintenance scheduling and routing problems. Most studies in the literature are concerned with finding generic maintenance feasible routes for the AMR. Feo & Bard [8] and Gopalan

Table 1
Analysis of articles published on related maintenance scheduling and routing problems

Reference	Covering Areas			Schedule		Maint. Considerations		
	FA	AMR	TA	CS	FH	CB	TB	COF
Feo & Bard [8]		√		√		√		√
Barnhart et al. [3]	√	√		√		√		
Gopalan & Talluri [10]		√		√		√		√
Moudani & Mora-Camino [20]		√	√		√		√	
Sriram & Haghani [27]			√	√		√		√
Sarac et al. [24]		√	√		√		√	
Liang et al. [16]		√		√		√		
Liang & Chaovalitwongse [15]	√	√		√		√		
Basdere & Bilge [4]		√	√		√	√		
Maher et al. [19]		√	√		√		√	
Liang et al. [17]		√	√	√		√		
Ruther et al. [22]		√	√		√		√	
Khaled et al. [12]		√	√		√	√		
Safaei & Jardine [23]		√	√		√		√	
This work			√		√		√	√

FA: Fleet Assignment, AMR: Aircraft Maintenance Routing, TA: Tail Assignment, CS: Cyclic Schedule, FH: Finite Horizon, CB: Check-Based, TB: Task-Based, COF: Compliant with Overall Framework.

& Talluri [9] have focused on solving the AMR by creating weekly cyclical flight schedules that spend the night at a maintenance station at least once every four days. An additional objective of Feo & Bard was to decrease the number of required maintenance stations. They both use a closed-loop city-day-network in which cities are represented by nodes, and the arcs between the nodes represent day-routes. A day-route contains all the flights an aircraft makes that day and is connected to the nodes that represent the cities from which the first flight of the day departs and at which the last flight of the day arrives. Their objectives are to find maintenance feasible cyclical schedules by connecting day-routes with each other and by making sure that the last flight of the last day-route arrives at the same station as the first flight from the first day-route. The advantage of a cyclical schedule (CS) is that, in theory, it could be executed in perpetuity. A disadvantage is that, in practice, using a single rotation is not applicable due to the stochastic nature of operations in the airline industry [4]. This stochastic nature is why operational models use a finite planning horizon (FH) to construct and assign the routes. Further, Liang & Chaovalitwongse [15] solve the integrated AMR and fleet assignment problem by using a compact time-space network.

Similar to Feo & Bard [8] and Gopalan & Talluri [9], Sriram & Haghani [27] use day-routes. These day-routes are taken as input and are used to solve the TA by creating ail-specific weekly cyclical flight routes. They use a matrix, representing the cost of maintenance per aircraft per city, to try and find a minimal cost route while adhering to the constraint of spending the night on a maintenance station at least once every four days.

A number of research studies have attempted to integrate the AMR and the TA. This integration is sometimes referred to in the literature as the operational aircraft maintenance routing problem (OAMR). Within these works a distinction can be made between papers that considered maintenance as a generic check required once every *d* days, and papers that considered a wider range of maintenance requirements. In Table 1 these two different maintenance considerations are denoted as check-based (CB) and task-based (TB). Considering maintenance requirements as a generic check, Basdere & Bilge [4] solve the OAMR over a weekly planning horizon. Their objective is to maximize the utilization of the total remaining flying time by planning the generic maintenance check as late as possible. Their approach considers all ground times as possibilities for maintenance. A drawback to their model is that aircraft can undergo maintenance at most once during the planning horizon. Khaled et al. [12] solve the generic check-based OAMR in a compact model, which allows them to operate a planning horizon longer than one week.

A number of papers presented OAMR models that take a wider range of maintenance tasks into account. Sarac et al. [24] and Maher et al. [19] model the daily OAMR. In Sarac et al. [24], if an aircraft is labeled high-time, which means that it might require maintenance, of any kind, at the end of the day, it is routed in such a way that it will spend the same night on a maintenance station. Moudani & Mora-Camino [20] solve an OAMR for a charter airline in which maintenance tasks are planned after the creation of the routes, in the resulting ground times. If no feasible solution can be found, it must be anticipated which flights should be delayed in order to create a feasible gap for the maintenance. The OAMR model proposed by Safaei & Jardine [23] attempts to minimize the mismatch between tail numbers' maintenance needs and maintenance opportunities using an iterative heuristic. Ruther et al. [22] solve an integrated OAMR and crew pairing four days before the day of operations.

Most research that has focused on solving the OAMR planned their routes for a one-week time horizon or shorter, while in practice the crew rosters are usually published roughly a month in advance. This means that in order to be able to stick with their rosters, crew members would most likely have to swap from aircraft after a flight numerous times. Necessitated crew-swapping increases the probability of extra delays since the delay of one aircraft would now propagate throughout the network. In fact, to avoid spreading delays in the network, crews should stay with the same aircraft as much as possible [6], or crews should only swap during extensively long ground times. The most common of these extensively long ground times is an overnight ground time. Therefore, swapping aircraft during an overnight ground time poses little risk of a propagated delay [17]. It is for this reason that our work limits its scope to the tail assignment in which the pre-constructed day-routes are to be partitioned over the tail numbers so that there are no costly crew swaps during the day.

The few research studies that did consider a wider range of maintenance requirements proposed very interesting models. However, in their current form it would leave airlines open to the mentioned risk of propagating delays. These papers are displayed in Table 1 as not compliant with an airline's overall planning framework (COF). It is also observed that apart from a case study by Senturk and Ozkol [25, 26], no research was found on the extent of the advantages that an individual task-based maintenance planning approach holds.

3. Problem definition

The tail assignment model in this thesis can be defined as follows. Given a flight schedule containing F flights, the objective of the model is to find a route for each tail number such that (a) all the flights are covered; (b) each tail number spends a night at a maintenance station at least once every d days to allow for necessary daily checks; (c) maintenance tasks are individually planned; (d) initial conditions for each tail number regarding location and maintenance needs are taken into account; (e) the process fits inside the airline's overall planning framework; (f) the cost of maintenance is minimized and (g) the maintenance station is given a capacity.

The tasks, that are to be planned individually, will be provided with both a due date and a remaining numbers of legal flight hours that can be flown before execution is mandatory. If either the due date or the remaining number of legal flight hours is exceeded before the task is executed, the aircraft loses its airworthiness. This means that whichever of these two limits appears first determines the cut-off point before which is a task must be executed.

The clear objective of most maintenance scheduling models is to minimize the cost of maintenance. However, because it is difficult to define and calculate a cost for maintenance often a surrogate objective is used [24]. If a maintenance task with positive remaining flying time is executed, then a portion of the flight capacity is wasted, and if necessary, parts that need to be replaced are done so before utilizing their useful lives [4]. To minimize this wasted flying time is equivalent to minimizing the unused legal flying times. Minimizing the unused flying time is taken as a surrogate objective for the cost minimization in this work, similar to Sarac et al. [24] and Basdere & Bilger [4]. The assumptions of our problem formulations are as follows:

1. The planned aircraft maintenance is only performed during the night
2. There is no aircraft operation during the night
3. The model considers a hub-and-spoke model where there is only one maintenance station
4. The given station capacity is fully available for the execution of the planned tasks

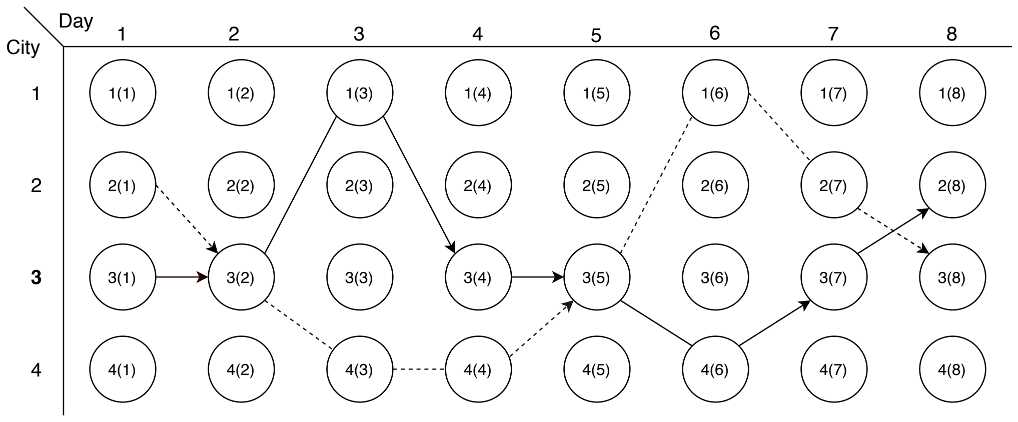


Figure 1: City-day network with a 7-day planning horizon

3.1. City-day network

Combining day-routes, as also done in the studies of Feo & Bard [8], Gopalan & Talluri [10], and Sriram & Haghani [27], is a compact and effective way of of constructing flight schedules for individual aircraft. Because any aircraft swaps that a crew might need to carry out are done during an overnight stay there is very little risk of propagating delays. Our presented model has a single maintenance station, and therefore instead of partitioning day-routes among tail numbers, we partition a combination of day-routes that start and/or end at the maintenance station. The combined day-routes are in this work referred to as *lines*. The lines are guaranteed to be shorter than the daily check interval of *d* days because they were created during the AMR where this is set as a strict requirement. A four cities, two aircraft, 7-day network is shown in Figure 1, in which the nodes represent the final destination city on a particular day and the arcs represent lines. In this example, city 3 represents the maintenance station. A closer look at the network in Figure 1 reveals that there are two nodes (3(2) and 3(5)) that have more than one incoming and outgoing line, and therefore at these nodes a decision must be made which aircraft to assign to which line.

3.2. Model formulation

In this section we propose a multi-commodity flow network model where each aircraft represents a separate commodity. The created lines in the AMR are taken as an input, and each line has an upper and lower capacity of one unit flow. Since it is not known beforehand how many flight hours each tail number will fly, it is also unknown which specific set of tasks will need to be planned within the time horizon. To account for this, the longest possible route for each tail number is determined. All the maintenance tasks that, at the start of the time horizon, have a remaining number of legal flight hours smaller than the number of flight hours in the longest route are included in the model. This does not mean that all tasks must be executed, because the actual traveled route may be shorter. This idea was also implemented by Bilge & Basdere [4], who labeled all aircraft with a check due in less remaining time than the longest possible route in the planning horizon as high-time aircraft. Our mathematical formulation is as follows:

$$Minimize \quad \sum_{i\in I}\sum_{j\in J}\sum_{t\in T_i} y_{itj} * S_{i,t} * min(Avg * (DD_{\mathrm{i,t}} - Date_{\mathrm{j}}), RFH_{\mathrm{i,t}} - \sum_{d=1}^{Date_{\mathrm{j}}}\sum_{j\in Js_{\mathrm{d}}} x_{\mathrm{ij}} * l_{\mathrm{j}}) \tag{1}$$

Subject to:

$$\sum_{i\in I} x_{ij} = 1 \qquad \forall j \in J \tag{2}$$

$$\sum_{j\in Ja_d} x_{ij} = 1 \qquad \forall i \in I, d = 2, 3, ..., n_{\mathrm{d}} \tag{3}$$

$$\sum_{j\in S_i} x_{ij} = 1 \qquad \forall i \in I \tag{4}$$

$$x_{ij} - \sum_{j\in C_j} x_{ij} \le 0 \qquad \forall i \in I, j \in J_b \tag{5}$$

$$\sum_{j\in J} y_{itj} \leq 1 \qquad \forall i \in I, t \in T_i \tag{6}$$

$$RFH_{i,t} - \sum_{j\in J} x_{ij} * l_j \geq \sum_{j\in J} y_{itj} * ms_j * -K \qquad \forall i \in I, t \in T_i \tag{7}$$

$$DD_{i,t} - n_d \geq \sum_{j\in J} y_{itj} * ms_j * -K \qquad \forall i \in I, t \in T_i \tag{8}$$

$$y_{itj} * (\sum_{d=1}^{Date_j} \sum_{j\in Js_d} x_{ij} * l_j) \leq RFH_{i,t} \qquad \forall i \in I, t \in T_i, j \in J \tag{9}$$

$$y_{itj} * Date_j \leq DD_{i,t} \qquad \forall i \in I, t \in T_i, j \in J \tag{10}$$

$$|T_i| * x_{ij} \geq \sum_{t\in T_i} y_{itj} \qquad \forall i \in I, j \in J \tag{11}$$

$$\sum_{i\in I} \sum_{j\in Je_d} \sum_{t\in T_i} y_{itj} * S_{i,t} \leq M_d \qquad d = 1, ..., n_d \tag{12}$$

$$x_{ij} \in \{0, 1\} \qquad \forall i \in I, j \in J \tag{13}$$

$$y_{itj} \in \{0, 1\} \qquad \forall i \in I, t \in T_i, j \in J \tag{14}$$

The objective function (1) is the minimization of the total unused flying time, where the size of the task is taken as a weight factor. The total unused flying time is calculated by multiplying the average number of flight hours an aircraft flies per day with the number of days a task is planned before its due date. A second way to calculate lost flying time is to subtract the remaining number of legal flight hours at the start of the planning horizon with the number of flight hours recorded before the task is sceduled. The minimum of these two values is selected as the true lost flying time. Constraints (2) are the flight coverage constraints. Constraints (3) limit each aircraft to fly at most 1 line at any given moment. Constraints (4) ensure that each aircraft is allocated a feasible starting line in terms of initial location and the first daily check. Constraints (5) are the flow constraints. Constraints (6) ensure that every task can be planned a maximum of one time within the planning horizon. Constraints (7) and (8) ensure that a task must be planned if its remaining legal flight hours are less than the constructed route for the aircraft, or if the due date of the task falls outside of the planning horizon. Constraints (9) and (10) ensure that *if* a task is planned it is planned before its remaining flight

Table 2
Parameters, sets and decision variables

Parameters	
n_p	number of aircraft in the fleet
n_d	number of days in the planning horizon
n_j	number of lines in the planning horizon
$n_{i,t}$	number of considered tasks in the planning horizon for AC i
l_j	length of of line j in flight hours
$DD_{i,t}$	due date of task t of AC i
$RFH_{i,t}$	remaining number of legal flight hours of task t of AC i at the start of the planning horizon
$Date_j$	end date of line j
$S_{i,t}$	size in man-hours of task t of AC i
ms_j	equals 1 if line j ends on the maintenance station, 0 otherwise
M_d	Capacity of the maintenance station in man-hours on day d
Avg	average number of recorded flight hours per day per aircraft
K	a sufficiently large number
Sets	
I	set of all aircraft
J :	set of all lines in the planning horizon
Ja_d	set of all lines active on day d
Js_d	set of all lines starting on day d
Je_d	set of all lines ending on day d
J_b	set of all lines except the ending lines
S_i	set of all possible starting lines for AC i
C_j	set of all connection lines of line j
T_i	set of considered tasks for AC i within the time horizon
Decision Variables	
x_{ij}	= 1 if aircraft i flies line j, 0 otherwise
y_{itj}	= 1 if aircraft i plans maintenance task $t \in T_i$ after line j, 0 otherwise

hours run out or before its due date is reached. Constraints (11) make sure that a task can only be planned on a night that the aircraft spends on the maintenance station. Constraints (12) are the capacity constraints of the maintenance station in man-hours. Constraints (13) and (14) are the integrality constraints on the decision variables.

In our formulation a maintenance tasks can be planned a maximum of one time within the given planning horizon. For most tasks this should cause no problem, however tasks with a very short interval could thus not be individually planned but would have to be grouped in the nearest daily check.

3.3. Model characteristics

To gain an appreciation for the model size, a breakdown of the problem sizes and computational results for different sets of parameter settings is presented in Table 3. The given tests are run with a planning horizon of 7 days and a maintenance capacity of 75 man-hours per night at a maintenance station. In these runs, the maintenance task sets for each aircraft consisted of either 10, or 20 tasks. The flight data for these runs was extracted from online sources tracking individual aircraft of fleets. In this case, the flight data of a set of Boeing 737s of a large European airline was used. Each aircraft has been given a unique set of tasks to be completed. The size of each task in man-hours is generated following a uniform distribution between 0.5 man-hours and 1.5 man-hours so that the average lies around 1 man-hour, which is the average size of a task in an A-check. Each task's due date is generated following a discrete uniform distribution. The lower bound of this distribution is the due date for the aircraft's first daily check, and the upper bound is the end of the planning horizon. The number of remaining legal flight hours for a task is generated following a discrete uniform distribution where the lower bound equals the number of flight-hours that are flown before the aircraft can its first daily check. This lower bound is necessary to guarantee the existence of a feasible solution. The upper bound is set at the average number of recorded flight hours per day multiplied by the number of days in the planning horizon. As the size of the model increases, so too naturally does the time it takes to solve the model. For fleets containing a large number of aircraft, it seems that the model becomes too large to solve with exact methods within a reasonable time. Therefore another solution method will have to be constructed to generate good solutions in a faster way.

Table 3
Computational results when solving the MIP, with $n_d = 7$

# AC	# Tasks	# Var	# Const	Residual Gap (%)	CPU (sec)	Obj value
5	10	1045	2281	0.00	1	258
	20	1995	4331	0.00	2	471
6	10	1452	3125	0.00	1	304
	20	2772	5945	0.00	7	607
7	10	1925	4099	0.00	5	385
	20	3675	7809	0.00	14	727
8	10	2464	5203	0.00	19	424
	20	4704	9923	0.00	44	863
9	10	3168	6636	0.00	53	434
	20	6048	12,666	0.00	110	913
10	10	4070	8464	0.00	632	432
	20	7770	16,164	0.00	2138	937
11	10	4840	10,024	0.00	1269	471
	20	9240	19,154	5.41	56,000	1085

4. Solution approach

Due to advances in hardware and exact solution methods, several MIP models can reach optimality within a reasonable amount of time or close to it [2]. This has led to a relatively new research area called matheuristics that attempts to combine exact algorithms with heuristics. In the literature, matheuristics have been proposed for several different routing problems. One of the main matheuristic approaches is the decomposition approach [2], where the problem is decomposed into smaller subproblems. Decomposition approaches are especially suitable for handling complex and interrelated problems [2], which could make it a good fit for our given model. The developed matheuristics in this study fall into two main categories: decomposition by aircraft matheuristics, which are presented in Section 4.1, and rolling horizon matheuristics, which are presented in Section 4.2.

4.1. Decomposition by aircraft matheuristics

In decomposition by aircraft matheuristics, the problem is decomposed into N subproblems, with N equalling the total number of aircraft. Each subproblem generates the route and maintenance schedule for a single tail number by solving the corresponding MIP to optimality using a branch and bound. For the first aircraft, all the lines and the full maintenance capacity are available. After the solution is found, the used lines are removed from the network, and the remaining maintenance capacity is updated. Subsequently, the MIP with the remaining lines and maintenance capacity is solved for the next aircraft until all aircraft are assigned a route and maintenance schedule. Two decomposition by aircraft matheuristics have been modeled, which are presented in the upcoming sections.

4.1.1. A depth first and random search combination

The decomposition by aircraft method presented in this section is inspired by the depth-first and random search heuristic used by Sriram & Haghani [27], which yielded good results for an aircraft maintenance routing problem that is similar to our own. In this matheuristic, a list of all considered aircraft is made in random order. The first aircraft of the list is selected, and the best possible route is constructed from all the available lines by solving the mathematical model presented in chapter 3 and adapted for a single aircraft. The available starting lines the aircraft has at its disposal are organized in a feasibility list, that contains the lines that it could feasibly choose as its first line. Lines that, if selected, would create feasibility problems for upcoming aircraft are excluded from the feasibility list and are thus not available for selection. For example, a line that is the only feasible starting line for an upcoming aircraft would not be included in the feasibility list. After the aircraft route is constructed, the used lines are removed from the network, and the feasibility list and remaining maintenance capacity per day for the maintenance station are updated. Subsequently, the second aircraft on the list is selected, and the process repeats itself until all the aircraft in the list are assigned routes. At the end of the process, a feasible solution is found, and all the solution values of the corresponding aircraft are summed to get the weighted value of the total lost flying time. This entire process is executed over multiple iterations, where the aircraft list is randomly shuffled for each iteration. If the found objective value of an iteration is lower than that of the previous best-found solution, the current iteration solution is saved as the current best solution. At the end of the matheuristic, the best-found solution is presented as the final solution. The pseudo-code of the depth first and random search combination is given in algorithm 1.

Algorithm 1: Depth First and Random Search Combination

```
k = 0,   n = 0;
K = Number of iterations;
N = Number of aircraft ;
while $k = k + 1 \leq K$ do
    1: Shuffle aircraft list;
    2: Reconstruct network and feasibility list and reset maintenance capacities;
    while $n = n + 1 \leq N$ do
        1: Select the nth aircraft on the aircraft list;
        2: Solve depth first MIP;
        3: Remove selected lines from network, and update the maintenance capacities and the feasibility list;
    end
    3: Save Solution if better than previous best;
end
```

4.1.2. Successive route checking
In the successive route checking matheuristic, similar to the depth-first and random search matheuristic, aircraft routes are created one by one. However, instead of constructing routes from the available lines, all the possible routes are created before solving the model. For every selected aircraft a very simple MIP is solved for all remaining possible routes, and subsequently the best route is selected. Because the route is given as input during each MIP, the routing constraints can be eliminated, and since the length of the route is known beforehand, the specific set of tasks that need to be scheduled can be determined before starting the MIP. The pseudo-code of the successive route checking matheuristic is given in algorithm 2.

Algorithm 2: Successive Route Checking

```
k = 0,   n = 0;
K = Number of iterations;
N = Number of aircraft ;
1: Determine all possible routes for each aircraft;
while $k = k + 1 \leq K$ do
    1: Shuffle aircraft list;
    2: Reset list of possible routes and the maintenance capacities, and reconstruct the feasibility list;
    while $n = n + 1 \leq N$ do
        1: Select the nth aircraft on the aircraft list;
        2: Successively solve simplified MIP for all remaining possible remaining routes and select the best
          route;
        3: Update remaining possible routes, the maintenance capacities and the feasibility list;
    end
    3: Save Solution if better than previous best;
end
```

4.2. Rolling horizon matheuristics
The matheuristic approach to decomposing a problem into time periods, or sub-horizons, is classified as a rolling horizon (mat)heuristic (RHM) by Archetti & Speranza [2]. Rolling horizon matheuristics have been applied for several transportation problems, such as Agra et al. [1], who used a rolling horizon matheuristic to solve an inventory routing problem in which ships are routed and scheduled between ports such that the demand for various fuel oil products is satisfied during the planning horizon. Rakke et al. [21] use a rolling horizon matheuristic to create a liquefied natural gas annual delivery program.

At the beginning of our RHM, all the tail numbers and all the lines that start from day 1 are considered. During the first iteration, a MIP is solved that distributes the starting lines as efficiently as possible and plans any required maintenance within the time period, with the goal of minimizing the total number of lost flying time. After the MIP is solved, the routing variables are frozen, and the sub-horizon shifts forwards. During the second iteration, all the tail numbers and all the lines that start on day 2 are considered. For the tail numbers, this consists of the aircraft that were assigned 1-day long lines during iteration 1. The pseudo-code of the RHM is presented in algorithm 3.

Algorithm 3: Rolling Horizon Matheuristic

```
k = 0;
U = Number of days in the planning horizon;
while $k = k + 1 \leq U$ do
    1: Select the set of tail numbers and lines starting from k;
    2: Determine the task set for each tail number that requires consideration;
    3: Solve the MIP, including feasibility constraint if k = 1;
    4: Freeze variables $x_{ij}$ from the current iteration;
end
```

The downside of an RHM, as implemented above, is that it, in a sense, is myopic. It only considers lines that start on the day of the iteration and does not take future lines that appear in an upcoming iteration into account. To counter this shortsightedness, a forecasting section can be included in the rolling horizon framework, similar to Rakke et al. [21] and Agra et al. [1]. Now every sub-horizon is decomposed into two time periods: the central period (CP) and the forecasting period (FP). In the central period, the MIP is solved, and the selected routes within it are frozen before moving to the next iteration. The forecasting period provides information to the central period about a larger part of the planning horizon. The idea is that by using a forecasting period, clearly sub-optimal solutions outside of the central period can be avoided so that the myopic nature of a rolling horizon matheuristic is mitigated [21].To provide the central period with information, a MIP is solved within the forecasting period, which often is a simplified version of the MIP in the central period. Figure 2 gives a graphical representation of the different time windows within a sub-horizon. Two main elements need to be considered for the forecasting period [21]:

1. A simplification strategy; and
2. The length of the forecasting period

A rolling horizon matheuristic that includes a forecasting section carries certain similarities with a well established family of control techniques known as Model Predictive Control (MPC) or Receding Horizon Control [5]. MPC is a model-based control approach, where a constrained optimization problem is solved over a given forecast horizon to determine a control action sequence for the controller. During every control step, only the first element of this action sequence is carried out. At the next time step the constrained optimization problem is reformulated and solved again, leading to a new control action sequence of which, again, only the first element is carried out, resulting in a receding horizon strategy. To reduce the computational burden of an MPC, Tian et al. [28] have proposed an Adaptive Control Resolution (ACR) approach. An ACR approach reduces the number of control variables by dividing their problem horizon in a number of phases with decreasing resolution as the phases become more distant in the future. It can be understood that the presented RHM with a forecasting section operates in a similar manner. During every iteration, only the first action (the flight schedule produced in the central period) is frozen or "carried out". Implementing a simplification strategy within the forecasting period is similar to implementing the ACR within the MPC, as it reduced the number of variables and is aimed to reduce the computation times.

Two forecasting strategies have been modeled and are presented in the following sections.

4.2.1. Exact Forecasting

Using an exact forecasting strategy means that there is no simplification for the forecasting period. The model in the forecasting period is subjected to the same constraints and objective function as in the central period. This means that the full model is solved for a time period that equals the length of the central period plus the length of the forecasting period, but only the lines that depart in the central period are frozen for the next iteration. It can logically be concluded that, if the length of the central period and the forecasting period together equals the length of the original problem's

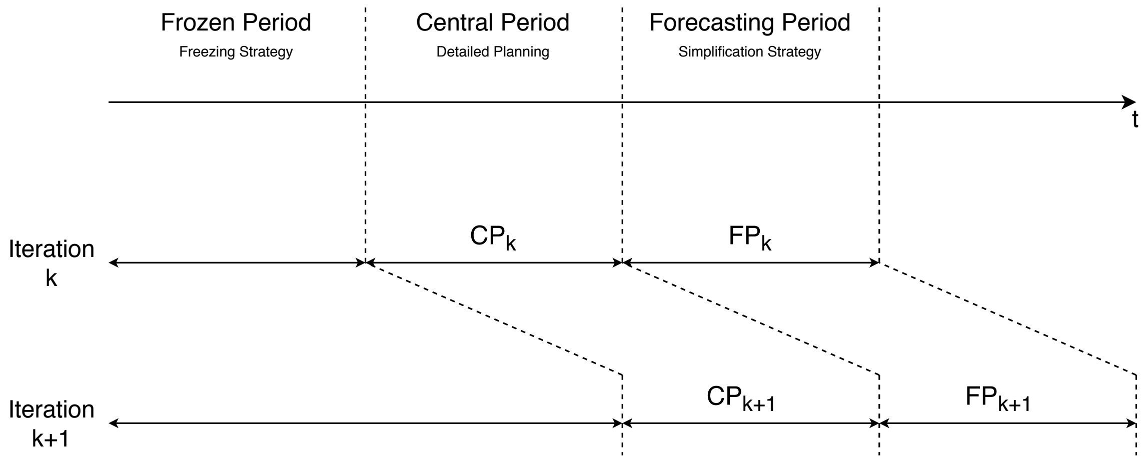


Figure 2: The different time-windows within the rolling horizon matheuristic using a forecasting approach [1]

planning horizon, the optimal solution is found. Algorithm 3 is updated to include the exact forecasting section and the pseudo-code is presented in algorithm 4.

It is no longer sufficient only to consider lines and aircraft departing from the maintenance station at the start of day k. Lines starting within the $\mathrm{FP_k}$ must also be considered. These lines need to be assigned either to aircraft that are operating previously frozen lines during the $\mathrm{CP_k}$, or to aircraft that are assigned a new line at the start of the $\mathrm{CP_k}$, which will finish before the end of the $\mathrm{FP_k}$, and thus will need to be assigned a second line within the considered time period. To allow for this, the flow constraints (5), have been adapted to the constraints (15).

$$x_{ij} - \sum_{j \in C_j} x_{ij} - \sum_{p \in P_j} z_{i,p} \leq 0 \qquad \forall i \in I, j \in Jm \tag{15}$$

Table 4
Added sets and decision variables for the exact forecasting

Sets	
Jm	Set of all active lines except for the starting lines
P_j	Set of all frozen lines that connect to line j
P_j	Set of all frozen lines that connect to line j
Decision Variables	
$z_{i,p}$	= 1 if aircraft i has flown frozen line p, 0 otherwise

Algorithm 4: Rolling Horizon Matheuristic With Exact Forecasting

```
k = 0;
U = Number of days in the planning horizon;
while $k = k + 1 \leq U$ do
    1: Identify the set of lines that start during the $\mathrm{CP_k}$ and $\mathrm{FP_k}$;
    2: Identify the set of tail numbers that will start a new line during the $\mathrm{CP_k}$ and $\mathrm{FP_k}$;
    3: Determine the task set for each tail number that requires consideration;
    4: Solve the mathematical model for the problem defined by $\mathrm{CP_k}$ and $\mathrm{FP_k}$;
    5: Freeze variables $x_{ij}$ in the central period $\mathrm{CP_k}$;
end
```

4.2.2. Average Flight Hour Forecasting

In this section, a strategy is presented that solves a simplified model for the forecasting period. As a first simplification, the capacity constraint of the maintenance station (equation 12) is removed. As a second simplification, the model will use the average number of flight hours per day (given as an input value) to predict the number of recorded flight hours for each tail number at the start and end of each line within the forecasting period. Based on these values, the model will assign a predicted tail-number dependent cost for each forecasted line by calculating the lost flying time that would be incurred due to maintenance tasks having to be planned early if this line is flown. The term presented in (16) is added to objective function (1) and introduces a new decision variable: q_{if}, which equals 1 if a line within the forecasting period is assigned to aircraft i. Constraints (17) and (18) are added to the model. The pseudo-code of the RHM with average flight hour forecasting is given in algorithm 5.

$$Minimize \quad \sum_{i \in I} \sum_{f \in F} q_{if} * Cost_{if} \tag{16}$$

$$\sum_{i \in I} q_{if} = 1 \qquad \forall f \in F \tag{17}$$

$$q_{if} - \sum_{j \in C_f} x_{ij} - \sum_{p \in P_f} z_{ip} - \sum_{f \in R_f} q_{if} \leq 0 \qquad \forall i \in I, f \in F \tag{18}$$

Table 5
Added sets and decision variables for the average flight hour forecasting

Sets	
$Cost_{i,f}$	Set that holds the predicted costs if future line f is assigned to aircraft i
F	Set of all lines starting in the forecasting period
C_f	Set of all lines, starting in the central period, that connect to line f
P_f	Set of all frozen lines that connect to line f
R_f	Set of all lines, starting in the forecasting period, that connect to line f
$z_{i,p}$	Set of binary values where a value equals 1 if aircraft i has flown frozen line p, 0 otherwise
Decision Variables	
$q_{i,f}$	= 1 if forecasted line f is assigned to aircraft i, 0 otherwise

Algorithm 5: Rolling Horizon Matheuristic With Average Flight Hour Forecasting

```
k = 0;
U = Number of days in the planning horizon;
while $k = k + 1 \leq U$ do
    1: Identify the set of lines that start during the $CP_k$ and $FP_k$;
    2: Identify the set of tail numbers that will start a new line during the $CP_k$ and $FP_k$;
    3: Determine the task set for each tail number that requires consideration;
    4: Calculate the predicted costs of the lines starting in the $FP_k$;
    5: Solve the mathematical model for the problem defined by $CP_k$ and $FP_k$;
    6: Freeze variables $x_{ij}$ in the central period $CP_k$;
end
```

4.3. Results

As could be observed from Table 3 exact results to our model were found for problem sets up to 11 aircraft and 10 tasks each. To verify and judge the quality of the solutions produced by our presented matheuristics, the matheuristic results will be compared to the results produced by the exact solutions for the tested problem sets for which the optimal solution was found. Each matheuristic has been run with the same data for the same problem sizes as presented in Table 3. The results are presented in Table 6. For each matheuristic and problem set the objective value of the solution and the required CPU time are denoted. The bottom row indicates the average optimality gap of all found objective values. The length of the forecasting period was extended up to two days for exact forecasting and up to six days for average flight hour forecasting. The results of the decomposition by aircraft matheuristics were found after completing ten iterations. This means that solutions have been found for ten different orders of the aircraft list. The total possible orders of the aircraft list equals the factorial of the number of aircraft included, which for all test cases vastly surpasses the number of ten. This means that only a small fraction of the solution space is explored. Greatly increasing the number of iterations is not feasible as this proportionally increases computation time. From these results, it is concluded that the rolling horizon matheuristics all present solutions with superior averaged objective values to the decomposition by aircraft methods. The RHMs with exact forecasting present the best solutions, but the required computation time rises steeply when increasing the forecasting period. It also appears that the solution results of the RHMs with average flight hour forecasting do not consequently improve when increasing the forecasting period. An explanation for this might be that the benefit of longer forecasting is offset by the decrease in prediction accuracy.

A second takeaway from the results presented in Table 6 is the fact that the decomposition by aircraft matheuristics take substantially longer to complete than the rolling horizon matheuristics (excluding the RHM with more than one day of exact forecasting) for a given of ten iterations. The explanation behind the long computation times of the depth first and random search matheuristic is that the first aircraft on the list of every iteration has a large number of lines to choose from, resulting in a MIP that still requires a substantial amount of time to solve. The successive route checking matheuristic needs an even longer computation time per iteration than the depth first random search combination, even though the MIPs solved in the SCRM are very elementary in nature. The reason for this is that the number of possible routes sharply rises with an increasing flight schedule size, and for the presented result the simplified MIP needed to be solved 6,530 times.

Table 6
Comparison of the objective values and solving times of the discussed matheuristics for large problem sets

		DFRS	SRCM	RHM NF	RHM EF-1	RHM EF-2	RHM AF-1	RHM AF-2	RHM AF-3	RHM AF-6
5 AC, 10 tasks	Obj. value	278	271	266	267	266	267	267	267	267
	CPU time (s)	7	11	1	2	2	1	2	2	2
5 AC, 20 tasks	Obj. value	511	510	487	471	471	471	471	471	471
	CPU time (s)	8	13	1	2	4	1	2	2	2
6 AC, 10 tasks	Obj. value	315	336	308	308	308	308	308	308	308
	CPU time (s)	8	15	1	2	4	1	2	3	3
6 AC, 20 tasks	Obj. value	675	652	607	607	607	607	607	607	607
	CPU time (s)	12	16	2	4	7	2	2	3	3
7 AC, 10 tasks	Obj. value	467	425	385	393	385	385	385	394	394
	CPU time (s)	11	21	1	2	5	1	3	3	3
7 AC, 20 tasks	Obj. value	802	807	731	741	727	731	731	731	731
	CPU time (s)	16	26	2	5	8	3	4	4	4
8 AC, 10 tasks	Obj. value	462	469	436	435	430	427	443	427	427
	CPU time (s)	12	38	1	5	10	2	3	3	4
8 AC, 20 tasks	Obj. value	945	972	868	867	867	868	885	885	885
	CPU time (s)	20	33	2	7	19	3	3	5	5
9 AC, 10 tasks	Obj. value	476	481	446	444	449	458	455	455	455
	CPU time (s)	18	63	2	7	25	2	4	4	6
9 AC, 20 tasks	Obj. value	1043	1071	939	958	939	950	948	950	950
	CPU time (s)	24	53	3	13	53	4	5	7	7
10 AC, 10 tasks	Obj. value	522	514	481	432	437	432	441	472	472
	CPU time (s)	24	150	2	16	46	3	5	6	7
10 AC, 20 tasks	Obj. value	1102	1103	997	965	962	977	1000	978	975
	CPU time (s)	48	127	3	26	250	4	6	8	9
11 AC, 10 tasks	Obj. value	563	572	620	534	486	553	550	536	536
	CPU time (s)	28	225	2	18	62	4	6	7	10
Average optimality gap		**12.5%**	**12.5%**	**5.1%**	**2.7%**	**1.5%**	**2.9%**	**3.6%**	**3.6%**	**3.6%**

DFRS: Depth-First Random Search, SRCM: Successive Route Checking Matheuristic, RHM NF: Rolling Horizon Matheuristic - No Forecasting, EF-1: Exact Forecasting For 1 Day, AF-1: Average Flight Hour Forecasting for 1 day.

The developed matheuristics were tested against larger problem sizes of up to 25 aircraft. To cope with the larger problem sets the capacity of the maintenance station was increased to 150 man-hours per night. The results of these tests are presented in Table 7. For these instances, as there were no exact solutions available, the presented gap does not equal the optimality gap but rather the gap with the best found objective value from all matheuristics. If the matheuristic was not yet completed after 3600 seconds, a dash is presented in Table 7.

Judging by this average gap, it seems clear that again the rolling horizon matheuristics outperform the decomposition by aircraft matheuristics in terms of objective values. Similar to the results of the smaller problem sets, the RHM with exact forecasting generates the best objective values for larger sets. However, for a forecasting horizon longer than one day, its computation time becomes too hefty for the largest problem sizes. The improvement in objective values in the smaller problem sets, generated by augmenting the RHM with the forecasting section based on average flight hours, seems to be reduced for larger sets. A forecasting length of two days, in fact, produced worse results than the RHM without forecasting.

Table 7
Comparison of the objective values and solving times of the discussed matheuristics for large problem sets

		DFRS	SRCM	RHM NF	RHM EF-1	RHM EF-2	RHM AF-1	RHM AF-2	RHM AF-3	RHM AF-6
15 AC, 10 tasks	Obj. value	797	816	746	671	643	665	765	731	717
	CPU time (s)	120	594	3	45	304	6	9	10	13
15 AC, 20 tasks	Obj. value	1903	1888	1692	1660	1547	1715	1640	1602	1616
	CPU time (s)	229	392	6	131	583	9	13	14	17
20 AC, 10 tasks	Obj. value	1192	1167	942	927	-	931	974	948	945
	CPU time (s)	232	2210	5	157	-	10	15	18	24
20 AC, 20 tasks	Obj. value	2641	2729	2280	2224	-	2278	2309	2215	2296
	CPU time (s)	485	1404	11	513	-	17	22	28	30
25 AC, 10 tasks	Obj. value	1450	-	1200	1149	-	1223	1184	1259	1187
	CPU time (s)	1570	-	8	413	-	15	22	35	36
25 AC, 20 tasks	Obj. value	-	-	3030	2898	-	2992	2973	2911	2865
	CPU time (s)	-	-	22	825	-	31	41	54	52
Average gap with best-found score		**24.2%**	**24.5%**	**6.7%**	**2.2%**	**0.0%**	**4.7%**	**6.9%**	**5.1%**	**4.1%**

DFRS: Depth-First Random Search, SRCM: Successive Route Checking Matheuristic, RHM NF: Rolling Horizon Matheuristic - No Forecasting, EF-1: Exact Forecasting For 1 Day, AF-1: Average Flight Hour Forecasting for 1 day.

5. Experimental study

In this section an experimental study will be performed with the aim of analyzing the potential impact of an individual task-based maintenance planning, as implemented by one of the presented matheuristics, on the airline industry. For this experimental study, the rolling horizon matheuristic with one day exact forecasting is selected as it has proven to produce good results in reasonable time for similar problem sizes that will be used in this experimental study. The results of the given matheuristic and that of a standard A-check scheduling strategy will be compared in terms of (1) total number of days aircraft have to be taken out of operations; and (2) total lost flying time. Furthermore, the distribution of required man-hours at the maintenance station per night will be compared.

In practice, it is likely that the routing and planning for the fleet would be updated at the end of each day to produce the new schedule for the newly considered planning horizon. To mimic this, the selected matheuristic will be transformed into a model with a running horizon. The model will be run before the start of every day for 60 days with a planning horizon of 7 days, and only the solution of the first day will be saved as the actually carried out flight and maintenance schedule.

For this experimental study, a fleet of 15 Boeing 737s of a large European airline has been considered. The flight data of these aircraft is extracted from a paid online aircraft-tracking service. Any gaps in the flight data are supplemented with fabricated data in line with the rest of the flight data. The interval of an A-check is put at 60 days, and the A-checks are spaced as evenly as possible within the given interval. An A-check is assumed to contain 100 individual maintenance tasks, with an average length of 1 man-hour per task. Half of these tasks are considered to have a shared due date with the set date of the A-check, and are thus considered to be part of a 'core-set' of tasks that return every A-check. The other fifty tasks are part of a set of tasks that just happen to be due within the 60-day interval between two consecutive A-checks, and are thus moved forward to be placed inside the first of these checks. For each aircraft, a unique set of these non-core tasks is generated. Furthermore, at the start of the interval, each non-core task is assigned a number of legal remaining flight hours the aircraft can operate before the task is due. The remaining number of legal flight hours are generated following a uniform distribution. However, putting the lower bound of this distribution at 0 flight hours could lead to an infeasible start of the model, because a task could be due before the aircraft has had a chance to visit a maintenance station for the first time. Therefore, the lower bound is set at a specific number of flight hours, after which the aircraft is guaranteed to have had the ability to visit a maintenance station. By examining the given lines at the start of the model, this number can be determined. The upper bound of the uniform distribution is determined by multiplying the number of average flight hours per day by 60. So in a standard A-check strategy all the non-core tasks would be moved up to the nearest A-check, whereas with our model they will be planned individually on overnight layovers on the maintenance station.

5.1. Results of the experimental study

This section will present the results of the experimental study. It must be noted that the presented lost flying hours are actually the *weighted* lost flying hours, as the size of a task is taken as a weight factor in the model. Since the average size of a task equals one man-hour, and the main interest is to examine the differences between the two methods, not the absolute sizes, we present it as the lost flying hours.

Figure 3 and Table 8 present the results on the lost flying days incurred during the experimental study for both tested approaches. The A-check planning requires every aircraft to be taken out of operations for a single day to complete the check. This means that for a fleet of 15 aircraft, a total of 15 lost flying days were incurred over the given interval. The proposed matheuristic was able to plan all the required maintenance tasks within overnight ground times at the maintenance station, which resulted in a total of 0 lost flying days.

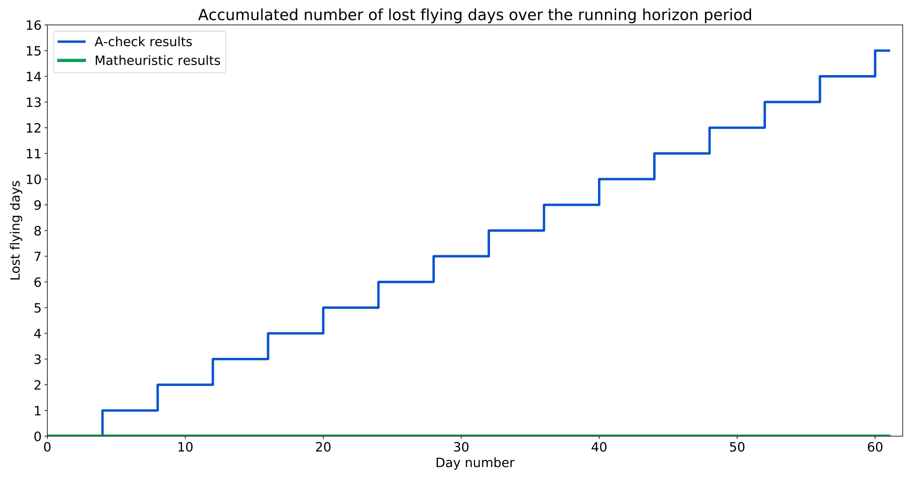


Figure 3: Visualization of the accumulation of lost flying days over the interval

Table 8
Comparison of the lost flying days incurred for each aircraft within the study between the A-check and the matheuristic

AC	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	**Total**
A-Check	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	**15**
Mat.	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**0**

AC: Aircraft, Mat.: Matheuristic

Figure 4 and Table 9 present the results on the lost flying hours incurred during the experimental study for both tested approaches. To determine the lost flying hours for the A-check strategy, an approximation had to be made, using the average flight hours per day. This means that if an A-check was performed on day 12, it is assumed that the aircraft had traveled a total number of flight hours at that point of 12 times the average flight hours per day. The lost flying hours of any task were then calculated by calculating the difference between their remaining number of legal flight hours at day 0 and the number of traveled flight hours on the day of the planned A-check. As seen in the results, there is a very substantial difference between the two methods in terms of lost flying hours. The number of 219.56 thousand lost flying hours, incurred with the A-check planning, decreased to 4.32 thousand lost flying hours when implementing the matheuristic, which equals a drop of over 98%. This drop effectively translates to a much more efficient maintenance planning, where tasks are performed when they are due, instead of when the latest A-check opportunity occurs.

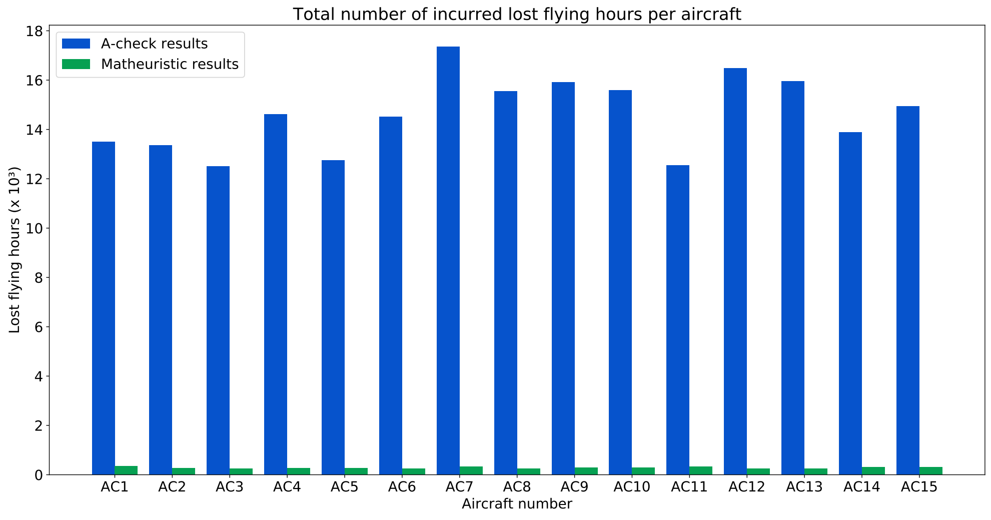


Figure 4: Visualization of the lost flying days incurred per aircraft

Table 9
Comparison of the lost flying days incurred for each aircraft within the study between the A-check and the matheuristic

AC	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	**Total**
A-Check	13.51	13.37	12.51	14.63	12.75	14.51	17.36	15.56	15.93	15.60	12.56	16.50	15.95	13.88	14.94	**219.56**
Mat.	0.36	0.28	0.25	0.27	0.27	0.26	0.33	0.25	0.30	0.29	0.34	0.26	0.26	0.31	0.30	**4.32**

AC: Aircraft, Mat.: Matheuristic

Figure 5 presents the results on the man-hours at the maintenance station for every night in the 60-day interval. Distinct peaks in workload can be seen for the A-check strategy. The results of the matheuristic show a more phased maintenance planning approach, as maintenance tasks are now be performed on overnight stays at the maintenance station. The still existing, but smaller, peaks in workload for the matheuristic are a result from the core set of tasks being due at the same moment.

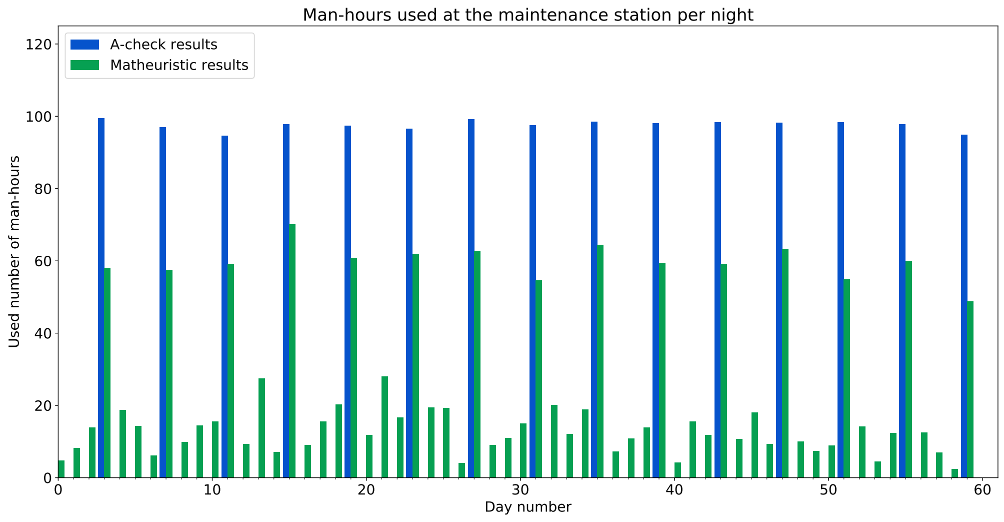


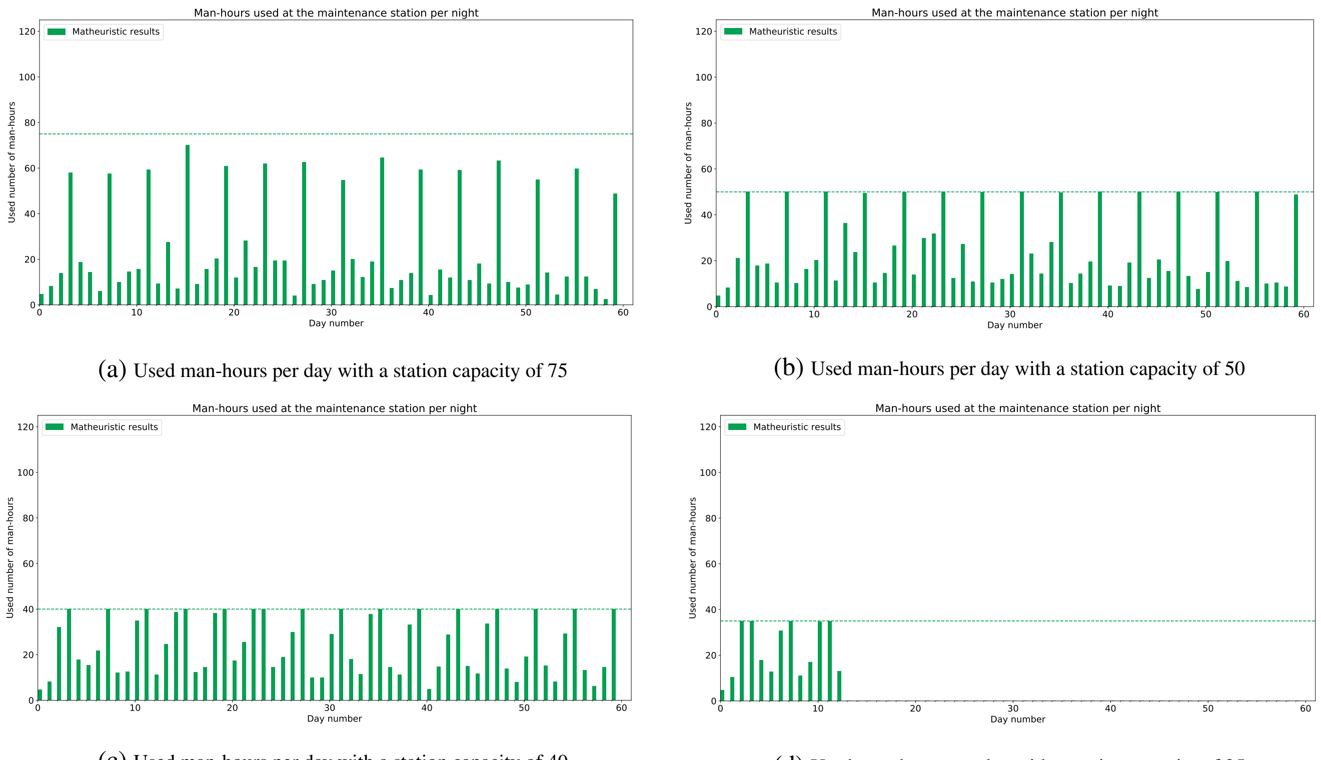
Figure 5: Visualization of the used man-hours at the maintenance station per day

5.2. Sensitivity analysis

A sensitivity analysis is conducted to examine how the capacity of the maintenance station affects the results of the selected matheuristic. Four different scenario's will be tested, with the capacity changing from 75, to 50, to 40, to 35 man-hours. The results of different scenarios in terms of lost flying hours are given in Table 10. The corresponding used man-hours per day at the maintenance station for each scenario are presented in Figure 6. The matheuristic was not able to find a feasible solution on the 13th day for a capacity of 35 man-hours. The results indicate a clear trend that increasing the station capacity has a positive effect on the incurred number of lost flying hours. In Figure 6a it is seen that on many occasions the matheuristic plans maintenance nights close to 75 man-hours. When decreasing the capacity a number of the tasks had to be moved forward, resulting in a larger number of lost flying hours. Decreasing the station capacity leads to a more phased maintenance planning, because the existence of peaks above a certain threshold is forbidden.

Table 10
Lost flying hours incurred with changing station capacity parameter (in man-hours)

Station Capacity	Lost Flying hours
75	4316
50	6178
40	7093
35	-



(a) Used man-hours per day with a station capacity of 75

(b) Used man-hours per day with a station capacity of 50

(c) Used man-hours per day with a station capacity of 40

(d) Used man-hours per day with a station capacity of 35

Figure 6: Visualization of the used man-hours per day at the maintenance station for varying station capacities

6. Impact on the airline industry

In the experimental study, the maintenance scheduling results of a standard A-check planning and the results of the rolling horizon matheuristic with one day exact forecasting were compared over an interval of 60 days for a fleet of 15 aircraft. Where the A-check planning required every aircraft to be taken out of operations for one day to complete the check, the matheuristic planned all required maintenance on overnight stays at the maintenance station. As a result, the aircraft did not have to be taken out of operations. This increase in aircraft availability can be used in multiple ways, and it is up to the airline to decide which way suits their needs the best. One straightforward way is to use this increased availability to expand the airline's flight schedule and operate more flights. Depending on the aircraft's utilization level, a day of operations may represent between \$75k and \$120k of additional revenue [7]. If the assumption is made that an aircraft currently requires 6 A-checks per year, the extra revenue could equal up to \$ 7.2 million per aircraft over a time period of 10 years. Further, this amount only reflects the potential savings on A-checks. Using an individual task-based strategy could possibly reduce the required downtime for C-checks as well.

Secondly, in the experimental study, the number of lost flying hours were compared between the standard A-check planning and our individual task based planning matheuristic. The number of lost flying hours decreased with over 98%, when using our proposed matheuristic, as opposed to the standard A-check planning. It is more difficult to put a price tag on the saved costs due to the decreased number of lost flying hours. However, as stated before, the better utilization of the maintenance tasks' intervals leads to less required maintenance in the long run. This could express itself in a smaller mechanics team that could handle the same fleet and could provide savings in salary costs.

Furthermore, the experimental study showed that a standard A-check planning is accompanied by distinct peaks in workload. Using an individual task-based maintenance planning results in a more phased maintenance approach which attenuates the peaks and creates a more balanced workload distribution.

A final aspect in which implementing our proposed model and solution method could impact the airline industry is the role of the maintenance planner. The planning of aircraft maintenance is often still a very manual process, relying on the experience of the planners. The combination of the planners' required experience and their importance to the day-to-day operations makes it difficult to recruit new employees for this job and has, in some cases, lead to an aging staff in this department. Implementing a system that can automate part of their job thus not only has the potential to save in salary costs but might actually prove instrumental to an airline's continuous operations in the long term.

In order to successfully implement our planning method there needs to be an increased flexibility in the roster scheduling of the mechanic workforce. The required man-hours at the maintenance station for a standard A-check strategy can be planned far in advance because the A-check are planned long before the day of operations. When implementing our model, the required man-hours at the maintenance station are much more variable and are only determined when solving the tail assignment. The tail assignment is solved much closer to the day of operations, which means that the mechanic workforce's schedules will be fixed closer to the day of operations.

7. Conclusion

In this study we propose a new aircraft routing and maintenance scheduling model that is able to plan tasks on an individual basis, whilst not leaving the airline exposed to the risk of propagating delays due to necessary crew swaps. The tasks that are planned individually are given both a due date and a remaining number of legal flight hours. The objective of the model is to utilize the remaining times of the individual tasks. Furthermore, the maintenance station is given a capacity in man-hours it has to its disposal to carry out the scheduled tasks. Several matheuristics were developed to solve the considered problem for larger sizes, because the computational burden for exact methods became too hefty. The created matheuristics are divided into two categories: decomposition by aircraft matheuristics and rolling horizon matheuristics. The analysis of our testing results showed that the rolling horizon matheuristics consistently outperformed the decomposition by aircraft matheuristics by some margin. To further improve the rolling horizon matheuristics, a forecasting section was introduced. Through exact forecasting excellent results were produced, however when increasing the problem size, or the length of the forecasting period beyond a certain size computation times too became quite long. When using average flight hour forecasting the results improved as well, however increasing the length of the forecasting period did not seem to yield a consistent improvement in results. Depending on the (1) the size of the fleet that an airline operates; (2) the number of tasks that they wish to plan on an individual basis; and (3) the available computing power, it is concluded that either the rolling horizon matheuristic with exact forecasting or with average flight hour forecasting is the best choice.

In an experimental study it was shown that implementing our model instead of a standard A-check planning can eliminate the need to take an aircraft out of operations, resulting in potential savings up to \$7.2 million per aircraft over a time period of 10 years. Furthermore, the number of lost flying hours, due to tasks being executed early, was reduced by over 98%, resulting in a much more efficient maintenance planning, and reducing the required maintenance over an aircraft's lifetime.

There are a number of improvements that can be made to our presented formulation. First, an overtime component could be added to the maintenance capacity. In some cases, it might be beneficial, or even necessary, to exceed the given man-hours of the station capacity. This could practically be done by letting mechanics work overtime or by deploying a larger mechanics team for that night. Planning overtime could be represented in the objective function by a penalty. A second improvement is to dive deeper into the nature of the different maintenance tasks. In practice, some maintenance tasks could have good synergy with each other, and therefore it might be beneficial to execute them together. Finally, the third recommendation for further research could be to include ground times during the day for maintenance planning as well. This would allow for more potential maintenance opportunities to be utilized. However, also considering ground times during the day as maintenance opportunities would increase the size and complexity of the model tremendously.

References

[1] Agra, A., Christiansen, M., Delgado, A., Simonetti, L., 2014. Hybrid heuristics for a short sea inventory routing problem. European Journal of Operational Research 236, 924–935.

[2] Archetti, C., Speranza, M.G., 2014. A survey on matheuristics for routing problems. EURO Journal on Computational Optimization 2, 223–246.

[3] Barnhart, C., Boland, N., Clarke, L., Johnson, E., Nemhauser, G., Shenoi, R., 1998. Flight string models for aircraft fleeting and routing. Transportation Science 32, 208–220.

[4] Basdere, M., Bilge, U., 2014. Operational aircraft maintenance routing problem with remaining time consideration. European Journal of Operational Research 235, 315–328.

[5] Camacho, E., Bordons, C., 2004. Model Predictive Control. volume 13.

[6] Cohn, A., Lapp, M., 2011. Airline Resource Scheduling. doi:10.1002/9780470400531.eorms0020.

[7] Deng, Q., Santos, B.F., Curran, R., 2019. A practical dynamic programming based methodology for aircraft maintenance check scheduling optimization. European Journal of Operational Research 281, 256–273.

[8] Feo, T., Bard, J., 1989. Flight scheduling and maintenance base planning. Management Science 35, 1415–1432.

[9] Gopalan, R., Talluri, K.T., 1998a. The aircraft maintenance routing problem. Operations research 46, 260–271.

[10] Gopalan, R., Talluri, K.T., 1998b. The aircraft maintenance routing problem. Operations research 46, 260–271.

[11] IATA's Maintenance Cost Technical Group, 2019. Airline Maintenance Cost Executive Commentary. Technical Report. International Air Transport Association.

[12] Khaled, O., Minoux, M., Mousseau, V., Michel, S., Ceugniet, X., 2017. A compact optimization model for the tail assignment problem. European Journal of Operational Research 264, 548–557.

[13] Klabjan, D., 2005. Large-Scale Models in the Airline Industry. pp. 163–196.

[14] Liang, Z., Chaovalitwongse, W., 2009. Optimization and Logistics Challenges in the Enterprise. Springer, Boston, MA. volume 30. chapter The Aircraft Maintenance Routing Problem. pp. 327–348.

[15] Liang, Z., Chaovalitwongse, W., 2013. A network-based model for the integrated weekly aircraft maintenance routing and fleet assignment problem. Transportation Science 47, 493–507.

[16] Liang, Z., Chaovalitwongse, W., Huang, H., Johnson, E., 2011. On a new rotation tour network model for aircraft maintenance routing problem. Transportation Science 45, 109–120.

[17] Liang, Z., Feng, Y., Zhang, X., Wu, T., Chaovalitwongse, W., 2015. Robust weekly aircraft maintenance routing problem and the extension to the tail assignment problem. Transportation Research Part B: Methodological 78, 238–259.

[18] Lieshout, R., Malighetti, P., Redondi, R., Burghouwt, G., 2015. The competitive landscape of air transport in europe. Journal of Transport Geography 50, 68–82.

[19] Maher, S.J., Desaulniers, G., Soumis, F., 2014. Recoverable robust single day aircraft maintenance routing problem. Computers and Operations Research 51, 130–145.

[20] Moudani, W., 2000. A dynamic approach for aircraft assignment and maintenance scheduling by airlines. Journal of Air Transport Management 6, 233–237.

[21] Rakke, J., Stålhane, M., Moe, C., Christiansen, M., Andersson, H., Fagerholt, K., Norstad, I., 2011. A rolling horizon heuristic for creating a liquefied natural gas annual delivery program. Transportation Research Part C Emerging Technologies 19, 896–911.

[22] Ruther, S., Boland, N., Engineer, F., Evans, I., 2016. Integrated aircraft routing, crew pairing, and tail assignment: Branch-and-price with many pricing problems. Transportation Science 51, 177–195.

[23] Safaei, N., Jardine, A., 2017. Aircraft routing with generalized maintenance constraints. Omega 80, 111–122.

[24] Sarac, A., Batta, R., Rump, C.M., 2006. A branch-and-price approach for operational aircraft maintenance routing. European Journal of Operational Research 175, 1850–1869.
[25] Senturk, C., Kavsaoglu, M.S., Nikbay, M., . Optimization of aircraft utilization by reducing scheduled maintenance downtime, in: 10th AIAA Aviation Technology, Integration, and Operations (ATIO) Conference.
[26] Senturk, C., Ozkol, I., 2018. The effects of the use of single task-oriented maintenance concept and more accurate letter check alternatives on the reduction of scheduled maintenance downtime of aircraft. International Journal of Mechanical Engineering and Robotics Research 7, 189–196.
[27] Sriram, C., Haghani, A., 2003. An optimization model for aircraft maintenance scheduling and re-assignment. Transportation Research Part A: Policy and Practice 37, 29–48.
[28] Tian, X., Negenborn, R., Overloop, P.J., Maestre, J., Sadowska, A., van de Giesen, N., 2017. Efficient multi-scenario model predictive control for water resources management with ensemble streamflow forecasts. Advances in Water Resources 109, 58–68.

B
Python Code - Exact Solver

```
# −*− coding: utf−8 −*−
"""
Created on Thu Apr 16 21:33:00 2020

@author: Alex
In this script the complete model formulation is solved using Gurobi
"""

import pandas as pd
from Line_extraction import Lines_Only #this is a function that extracts line data from
    flight data
from gurobipy import *

TaskData = pd.read_excel('TaskData_20_tasks.xlsx', sheet_name=None)
FlightData = pd.read_excel('FlightData.xlsx', sheet_name=None)
TailNumbers = pd.ExcelFile('FlightData.xlsx').sheet_names
TailNumbers = ['PH−BGF','PH−BGG','PH−BGH','PH−BGI','PH−BGK'] #this list contains all the tail
     numbers that are considered for the problem
#Parameters:
Planning_horizon = 7
StartingDay = 43831 #Excel uses a number for a date, which works very well. 43831 translates
    to 1−1−2020
Station_Capacity = 150 #in man−hours per night

#extract the lines from the flightdata. Within the function Lines_Only also a due date for
    the first standard check is randomly generated
Lines, StartLines, EndLines,standard_check_due_date = Lines_Only(FlightData,TailNumbers)
#calculate the average flight hours flown per day
average_FH_per_day = round(sum([Lines[b]['Flight Hours'] for b in Lines])/(len(TailNumbers)*
    Planning_horizon),1)

#%% Preprocessing for building the routes
#Here multiple dictionaries are built containing line information, which are used when
    defining the model
Lines_starting_each_day = {}
Lines_ending_each_day = {}
Lines_active_each_day = {}
for i in range(Planning_horizon):
    for j in Lines:
       Lines_starting_each_day[StartingDay+i] = [a for a in Lines if Lines[a]['StartDate'] ==
            StartingDay+i]
       Lines_ending_each_day[StartingDay+i] = [a for a in Lines if Lines[a]['EndDate'] ==
           StartingDay+i]
       Lines_active_each_day[StartingDay+i] = [a for a in Lines if Lines[a]['StartDate'] <=
           StartingDay+i and Lines[a]['EndDate'] >= StartingDay+i]

#It is necessary to build a dictionary containing all the possible connections. This
    dictionary will be used in the flow constraint
Connections = {}
for i in Lines:
    Connections[i] = []
    ThisLOF = Lines[i]
    EndDateThisLOF = ThisLOF['EndDate']
    for j in Lines:
        if Lines[j]['StartDate'] == EndDateThisLOF+1:
            Connections[i].append(j)
for i in EndLines: #the connections of the final lines fall outside of our planning horizon
    Connections.pop(i)

#In the given flight data Amsterdam is the maintenance station
#Whether or not Lines end on amsterdam. this is necessary for the ending lines, which could
    end on different stations than Amsterdam.
MS = {} #MS = maintenance station
for j in Lines:
    if Lines[j]['Arrival'] == 'Amsterdam':
        MS[j] = 1
    else:
        MS[j] = 0

#%% Find all possible routes and their lengths, this is needed to dermine which tasks should
```

```
    be considered
routes = {}
routeslength= {}
RouteCounter = 0
for i in StartLines:
    for i2 in Connections[i]:
        if i2 in EndLines:
            route = [i,i2]
            routes[RouteCounter] = route
            routeslength[RouteCounter] = sum([Lines[b]['Flight Hours'] for b in route])
            RouteCounter += 1
        else:
            for i3 in Connections[i2]:
                if i3 in EndLines:
                    route =[i,i2,i3]
                    routes[RouteCounter] = route
                    routeslength[RouteCounter] = sum([Lines[b]['Flight Hours'] for b in route
                        ])
                    RouteCounter += 1
                else:
                    for i4 in Connections[i3]:
                        if i4 in EndLines:
                            route =[i,i2,i3,i4]
                            routes[RouteCounter] = route
                            routeslength[RouteCounter] = sum([Lines[b]['Flight Hours'] for b
                                in route])
                            RouteCounter += 1
                        else:
                            for i5 in Connections[i4]:
                                if i5 in EndLines:
                                    route =[i,i2,i3,i4,i5]
                                    routes[RouteCounter] = route
                                    routeslength[RouteCounter] = sum([Lines[b]['Flight Hours'
                                        ] for b in route])
                                    RouteCounter += 1
                                else:
                                    for i6 in Connections[i5]:
                                        if i6 in EndLines:
                                            route =[i,i2,i3,i4,i5,i6]
                                            routes[RouteCounter] = route
                                            routeslength[RouteCounter] = sum([Lines[b]['
                                                Flight Hours'] for b in route])
                                            RouteCounter += 1
                                        else:
                                            for i7 in Connections[i6]:
                                                if i7 in EndLines:
                                                    route =[i,i2,i3,i4,i5,i6,i7]
                                                    routes[RouteCounter] = route
                                                    routeslength[RouteCounter] = sum([Lines[b
                                                        ]['Flight Hours'] for b in route])
                                                    RouteCounter += 1
#Determine the starting points for each AC
StartPoint = {}
for i in range(len(TailNumbers)):
    Starter = StartLines[i]
    #the first Startline is from the first aircraft and so on
    StartPoint[TailNumbers[i]] = Lines[Starter]['Departure']
#select the routes for each AC that start at the same station as the AC and determine which
    of these routes is the longest
max_possible_length= {}
route_selection = {}
for i in TailNumbers:
    route_selection[i] = {}
    for j in routes:
        if Lines[routes[j][0]]['Departure'] == StartPoint[i]:
            route_selection[i][j] = routes[j]
    max_possible_length[i] = max([routeslength[b] for b in route_selection[i]])

#Determine the set of tasks for each AC that need to be considered
#We consider the AC to have travelled 0 FH at the start of the time horizon
Tasks = {}
```

```
for TN in TailNumbers:
    Tasks_per_tail = {}
    for index, row in TaskData[TN].iterrows():
        #Should the task be considered in this time horizon? --> check the amount of FH in
            the route
        DueDate = row['Task Due']
        DueFH = row['FH Due']
        if DueDate <= (StartingDay + Planning_horizon) and DueDate > StartingDay or DueFH <=
            max_possible_length[TN]:
            Task = {'Task#':index,
                    'TaskMH': row['Task MH'],
                    'Task Due Date' : row['Task Due'],
                    'Task FH Due' : row['FH Due']
                }
            Tasks_per_tail[index]=Task
    Tasks[TN] = Tasks_per_tail

#Now from the startpoint determine the set of 1st day Lines possible for each tail number
PossibleStartLines = {}
for i in TailNumbers:
    PossibleStartLines[i] = [j for j in StartLines if Lines[j]['Departure'] == StartPoint[i]
        and Lines[j]['EndDate'] < standard_check_due_date[i]]

#%%

m = Model('Exact_Solver')

My_X_Variables = tuplelist([])
for i in range(len(TailNumbers)):
    for j in Lines:
        My_X_Variables.append((i,j))
My_Y_Variables = tuplelist([])
for i in range(len(TailNumbers)):
    for t in Tasks[TailNumbers[i]]:
        for j in Lines:
            My_Y_Variables.append((i,t,j))

#the following help variables are used to calculate the number of lost flying hours. This was
     necessary because a MIN or MAX function only works in a constraint and not in an
    objective function in Gurobi
LFH = {}
LFH2 = {}
LFH3 = {}
for i in range(len(TailNumbers)):
    for t in Tasks[TailNumbers[i]]:
        for j in Lines:
            LFH[i,t,j] = m.addVar(lb = -1000, ub = 1000)
            LFH2[i,t,j] = m.addVar(lb = -1000, ub = 1000)
            LFH3[i,t,j] = m.addVar(lb = -1000, ub = 1000)

x = m.addVars(My_X_Variables, vtype=GRB.BINARY, name='x')   # equals 1 if AC i flies line j,
    0 otherwise
y = m.addVars(My_Y_Variables, vtype=GRB.BINARY, name='y')   # equals 1 if AC i plans task t
    AFTER line j, 0 otherwise

#minimize total number of lost flying hours
m.setObjective(quicksum(y[i,t,j]*Tasks[TailNumbers[i]][t]['TaskMH']*LFH3[i,t,j] for i in
    range(len(TailNumbers)) for j in Lines for t in Tasks[TailNumbers[i]]),GRB.MINIMIZE)

#------ Routing Section

#Every line must be flown by an AC:
m.addConstrs((quicksum(x[i,j] for i in range(len(TailNumbers))) == 1 for j in Lines),name='c1
    ')

#Every AC can fly at most one line per day. This should exclude the first day because there
    is a separate constraint for that one:
for d in range(StartingDay+1,StartingDay+Planning_horizon):
    m.addConstrs(quicksum(x[i,j] for j in Lines_active_each_day[d] ) <= 1 for i in range(len(
        TailNumbers)))
```

```
#The first line of an AC should start from its initial location:
m.addConstrs(quicksum(x[i,j] for j in PossibleStartLines[TailNumbers[i]]) == 1 for i in range
    (len(TailNumbers)))

#If AC i flies line j, then it also must fly a connecting line, this excludes the last day
m.addConstrs(x[i,j] - quicksum(x[i,p] for p in Connections[j]) <= 0 for i in range(len(
    TailNumbers)) for j in Connections)

#----- Task Planning

#Every task can at most be planned once:
m.addConstrs(quicksum(y[i,t,j] for j in Lines) <= 1 for i in range(len(TailNumbers)) for t in
     Tasks[TailNumbers[i]])

#All tasks in T_i which are due before a due date should be planned if that due date is
    closer than the planning horizon
m.addConstrs(-1000*quicksum(y[i,t,j]*MS[j] for j in Lines) <= Tasks[TailNumbers[i]][t]['Task
    Due Date'] - (StartingDay+Planning_horizon+1) for i in range(len(TailNumbers)) for t in
    Tasks[TailNumbers[i]])

#All tasks in T_i which are due before X flight hours should be planned if the constructed
    route is longer than X flight hours.
m.addConstrs(-1000*quicksum(y[i,t,j]*MS[j] for j in Lines) <= (Tasks[TailNumbers[i]][t]['Task
     FH Due'] - quicksum(x[i,j]*Lines[j]['Flight Hours'] for j in Lines)) for i in range(len(
    TailNumbers)) for t in Tasks[TailNumbers[i]])

#If task t of AC i is scheduled, it must be scheduled before the due date:
m.addConstrs(y[i,t,j]*Lines[j]['EndDate']+1 <= Tasks[TailNumbers[i]][t]['Task Due Date'] for
    i in range(len(TailNumbers)) for t in Tasks[TailNumbers[i]] for j in Lines)

#If task t of AC i is scheduled, it must be scheduled before the due flight hours:
m.addConstrs(y[i,t,j]* (Lines[j]['Flight Hours'] + quicksum(x[i,p]*Lines[p]['Flight Hours']
    for d in range(StartingDay,Lines[j]['StartDate']) for p in Lines_starting_each_day[d]))
    <= Tasks[TailNumbers[i]][t]['Task FH Due'] for i in range(len(TailNumbers)) for t in
    Tasks[TailNumbers[i]] for j in Lines)

#A task t of AC i can only be after a line that is flown by AC i:
m.addConstrs(len(Tasks[TailNumbers[i]])*x[i,j] >= quicksum(y[i,t,j] for t in Tasks[
    TailNumbers[i]]) for i in range(len(TailNumbers)) for j in Lines)

#The capacity constraint
for d in range(StartingDay,StartingDay+Planning_horizon):
    m.addConstr(quicksum(y[i,t,j]*Tasks[TailNumbers[i]][t]['TaskMH'] for i in range(len(
        TailNumbers)) for j in Lines_ending_each_day[d] for t in Tasks[TailNumbers[i]]) <=
        Station_Capacity)

#Help variable for the objective function:
for i in range(len(TailNumbers)):
    for j in Lines:
        for t in Tasks[TailNumbers[i]]:
            m.addConstr(LFH[i,t,j] == Tasks[TailNumbers[i]][t]['Task FH Due']- (Lines[j]['
                Flight Hours'] + quicksum(x[i,k]*Lines[k]['Flight Hours'] for d in range(
                StartingDay,Lines[j]['StartDate']) for k in Lines_starting_each_day[d]))) #
                lost flying hours due to task being executed before remaining legal FH
            m.addConstr(LFH2[i,t,j] == average_FH_per_day*(Tasks[TailNumbers[i]][t]['Task Due
                 Date'] -(Lines[j]['EndDate']+1))) #Lost flying hours due to task being
                executed before its due date
            m.addConstr(LFH3[i,t,j] == min_(LFH[i,t,j],LFH2[i,t,j]))

m.optimize()

line_solution = m.getAttr('x', x)
task_solution = m.getAttr('x', y)
Flying_Schedule = [(i,j) for (i,j) in line_solution if line_solution[i,j] > 0.9]
Maintenance_Schedule = [(i,t,j) for (i,t,j) in task_solution if task_solution[i,t,j] > 0.9]

#print the solutions
print('Number of AC: ' + str(len(TailNumbers)))
print('Score :' +str(m.objval))
```

C
Python Code - Depth First and Random Search

```
# -*- coding: utf-8 -*-
"""
Created on Thu Apr 16 21:33:00 2020

@author: Alex
In this script i run a depth first and random search matheuristic to solve the given problem.
"""

import pandas as pd
from Line_extraction import Lines_Only
from gurobipy import *
import random
import copy
import time

TaskData = pd.read_excel('TaskData_20_tasks.xlsx', sheet_name=None)
FlightData = pd.read_excel('FlightData.xlsx', sheet_name=None)
TailNumbers = ['PH-BGF','PH-BGG','PH-BGH','PH-BGI','PH-BGK'] #this list contains all the tail
     numbers that are considered for the problem

#Parameters
Planning_horizon = 7
StartingDay = 43831 #Excel uses a number for a date, which works very well. 43831 translates
    to 1-1-2020
Station_Capacity = 150 #in man-hours per night

Maint_capacity = {}
for i in range(StartingDay,StartingDay+Planning_horizon):
    Maint_capacity[i] = Station_Capacity

#extract the lines from the flightdata. Within the function Lines_Only also a due date for
    the first standard check is randomly generated
Lines, StartLines, EndLines,standard_check_due_date = Lines_Only(FlightData,TailNumbers)
#calculate the average flight hours flown per day
average_FH_per_day = round(sum([Lines[b]['Flight Hours'] for b in Lines])/(len(TailNumbers)*
    Planning_horizon),1)

#%% Preprocessing for building the routes
#Here multiple dictionaries are built containing line information, which are used when
    defining the model
Lines_starting_each_day = {}
Lines_ending_each_day = {}
Lines_active_each_day = {}
for i in range(Planning_horizon):
    for j in Lines:
        Lines_starting_each_day[StartingDay+i] = [a for a in Lines if Lines[a]['StartDate'] ==
             StartingDay+i]
        Lines_ending_each_day[StartingDay+i] = [a for a in Lines if Lines[a]['EndDate'] ==
            StartingDay+i]
        Lines_active_each_day[StartingDay+i] = [a for a in Lines if Lines[a]['StartDate'] <=
            StartingDay+i and Lines[a]['EndDate'] >= StartingDay+i]

#It is necessary to build a dictionary containing all the possible connections. This
    dictionary will be used in the flow constraint
Connections = {}
for i in Lines:
    Connections[i] = []
    ThisLOF = Lines[i]
    EndDateThisLOF = ThisLOF['EndDate']
    for j in Lines:
        if Lines[j]['StartDate'] == EndDateThisLOF+1:
            Connections[i].append(j)
for i in EndLines: #the connections of the final lines fall outside of our planning horizon
    Connections.pop(i)

#In the given flight data Amsterdam is the maintenance station
#Whether or not Lines end on amsterdam. this is necessary for the ending lines, which could
    end on different stations than Amsterdam.
MS = {} #MS = maintenance station
for j in Lines:
    if Lines[j]['Arrival'] == 'Amsterdam':
        MS[j] = 1
    else:
        MS[j] = 0

# Find all possible routes and their lengths, this is needed to dermine which tasks should be
     considered
routes = {}
routeslength= {}
RouteCounter = 0
for i in StartLines:
    for i2 in Connections[i]:
        if i2 in EndLines:
            route = [i,i2]
            routes[RouteCounter] = route
            routeslength[RouteCounter] = sum([Lines[b]['Flight Hours'] for b in route])
            RouteCounter += 1
        else:
            for i3 in Connections[i2]:
                if i3 in EndLines:
                    route =[i,i2,i3]
                    routes[RouteCounter] = route
                    routeslength[RouteCounter] = sum([Lines[b]['Flight Hours'] for b in route
                        ])
                    RouteCounter += 1
                else:
                    for i4 in Connections[i3]:
                        if i4 in EndLines:
                            route =[i,i2,i3,i4]
                            routes[RouteCounter] = route
                            routeslength[RouteCounter] = sum([Lines[b]['Flight Hours'] for b
                                in route])
                            RouteCounter += 1
                        else:
                            for i5 in Connections[i4]:
                                if i5 in EndLines:
                                    route =[i,i2,i3,i4,i5]
                                    routes[RouteCounter] = route
                                    routeslength[RouteCounter] = sum([Lines[b]['Flight Hours'
                                        ] for b in route])
                                    RouteCounter += 1
                                else:
                                    for i6 in Connections[i5]:
                                        if i6 in EndLines:
                                            route =[i,i2,i3,i4,i5,i6]
                                            routes[RouteCounter] = route
                                            routeslength[RouteCounter] = sum([Lines[b]['
                                                Flight Hours'] for b in route])
                                            RouteCounter += 1
                                        else:
                                            for i7 in Connections[i6]:
                                                if i7 in EndLines:
                                                    route =[i,i2,i3,i4,i5,i6,i7]
                                                    routes[RouteCounter] = route
                                                    routeslength[RouteCounter] = sum([Lines[b
                                                        ]['Flight Hours'] for b in route])
                                                    RouteCounter += 1

#Determine the starting points for each tail
StartPoint = {}
for i in range(len(TailNumbers)):
    Starter = StartLines[i]
    #the first Startline is from the first aircraft and so on
    StartPoint[TailNumbers[i]] = Lines[Starter]['Departure']
#select the routes for each AC that start at the same station as the AC and determine which
    of these routes is the longest
max_possible_length= {}
route_selection = {}
for i in TailNumbers:
    route_selection[i] = {}
    for j in routes:
        if Lines[routes[j][0]]['Departure'] == StartPoint[i]:
            route_selection[i][j] = routes[j]
    max_possible_length[i] = max([routeslength[b] for b in route_selection[i]])

#Determine the set of tasks for each AC that need to be considered
#We consider the AC to have travelled 0 FH at the start of the time horizon
Tasks = {}
for TN in TailNumbers:
    Tasks_per_tail = {}
    for index, row in TaskData[TN].iterrows():
        #Should the task be considered in this time horizon? --> check the amount of FH in
            the route
        DueDate = row['Task Due']
        DueFH = row['FH Due']
        if DueDate <= (StartingDay + Planning_horizon) and DueDate > StartingDay or DueFH <=
            max_possible_length[TN]:
            Task = {'Task#':index,
                    'TaskMH': row['Task MH'],
                    'Task Due Date' : row['Task Due'],
                    'Task FH Due' : row['FH Due']
                }
            Tasks_per_tail[index]=Task
    Tasks[TN] = Tasks_per_tail

#Now from the startpoint determine the set of 1st day Lines possible for each AC, also the
    first task due date of a tailsign needs to be taken into account
OptionalStartLines = {}
for i in TailNumbers:
    OptionalStartLines[i] = []
    #find earliest task due by date
    thisdue = StartingDay + Planning_horizon
    for p in Tasks[i]:
        if Tasks[i][p]['Task Due Date'] < thisdue:
            thisdue = Tasks[i][p]['Task Due Date']
    #find the earliest task by due FH
    thisdueFH = 5000 #random large number
    for p in Tasks[i]:
        if Tasks[i][p]['Task FH Due'] < thisdueFH:
            thisdueFH = Tasks[i][p]['Task FH Due']
    for j in StartLines:
        if Lines[j]['Departure'] == StartPoint[i]: #if the line starts from the same station
            as the AC
            if Lines[j]['EndDate'] < standard_check_due_date[i]: #if the line ends before the
                standard check due date
                if Lines[j]['EndDate'] < thisdue:       #if the line ends before the earliest
                    due date of a task
                    if Lines[j]['Flight Hours'] <= thisdueFH:   #if the line ends before the
                        remaining legal FH of the nearest task are out
                        OptionalStartLines[i].append(j) #if all these conditions are met then
                             it is an optional start line

#%% Building the feasibility list
feasibility_list = {}
for i in TailNumbers:
    feasibility_list[i] = []
    for j in OptionalStartLines[i]:
        feasibility_list[i].append(j) #we start off by saying that every line from the

#if 1 AC has only 1 possible start LOF and 1 other AC has 2, but 1 of those is that singular
    possible startLOF for the other, than AC 2 also only has 1 possible start
k = 0
reserved_list=[]
while k == 0:
    k = 1
    for i in feasibility_list: #for every AC
        for p in feasibility_list[i]: #for every one of the feasible starting lines for the
            considered AC
            if p in reserved_list:  #if this starting line is reserved for another AC
                if len(feasibility_list[i]) > 1: #if there is more than 1 feasible start line
                    remaining
                    feasibility_list[i].remove(p)   #then remove it as a feasible line
                    k=0
        if len(feasibility_list[i]) == 1: #if there is only 1 feasible line left for the AC
            under consideration
            if feasibility_list[i][0] not in reserved_list: #and its not reserved by another
                AC
                reserved_list.append(feasibility_list[i][0])    ##then reserve it for this AC

#%% Begin the heuristic
number_of_iterations = 10
Current_Solution_Score = 1000000 #arbitrary large number
ScoreTracker = {}
ListTracker = {}
i = int(time.time()*1000) #random seed based on current time, to always have random aircraft
    list
random.seed(i)
stopwatch_start = time.time() #start a timer
for iteration in range(number_of_iterations):
    Flying_Schedule = {}
    Maintenance_Schedule = {}
    objective_score = {}
    #Building the aircraft list
    aircraft_list = random.sample(TailNumbers,len(TailNumbers))
    #Because within a decomposition by aircraft method, not all routes are created at the
        same time, we refer to the remaining lines (that have not been picked by a previous
        AC) as 'active'
    #Import the active line list
    active_Lines = copy.deepcopy(Lines) #before each iteration we start of with all the lines
         available
    active_Maint_capacity = copy.deepcopy(Maint_capacity) #before each iteration we start of
        with all remaining maintenance capacity available
    active_feasibility_list = copy.deepcopy(feasibility_list) #the feasibility list will have
         to be updated throughout the heuristics as well

    for n in aircraft_list: #start assigning routes to the aircraft one by one

        m = Model('Depth_First_Random_Search')
        aircraftID = [TailNumbers.index(n)] #the aircraftID is the position of the selected
            AC in the TailNumbers list. By using this we can always refer to the correct AC-
            specific data

        #Not all lines will be available for the current AC because of reservations in the
            feasibility list
        #first remove current AC from feasibility list
        active_feasibility_list.pop(n) #a dictionary uses pop(), not remove()
        possible_Lines = copy.deepcopy(active_Lines) #possible lines refers to the lines that
             the AC is allowed to choose
        for it in active_feasibility_list:
            if len(active_feasibility_list[it]) <= 1: #if there is only 1 feasible starting
                line remaining for a certain AC, then the current AC may not choose this line
                possible_Lines.pop(active_feasibility_list[it][0])
        #Sometimes a 'deadlock' can occur: 2 AC both have the same 2 possible Lines, then
            these 2 Lines must be reserved for these AC, if another AC takes one of these the
             result will be infeasible
        #Thats why we have to spot and remove deadlocks:
        deadlock_Lines = []
        k = 0
        while k == 0:  #one deadlock being removed can lead to another deadlock appearing,
            thats why we have to iterate until no more deadlocks can be found
            k=1
            for TN in active_feasibility_list:  #for each tail number (TN) remaining in the
                feasibility list
                if len(active_feasibility_list[TN]) > 1:
                    options1 = active_feasibility_list[TN] # a list of the feasible lines for
                         the TN
                    number_of_duplicates = 0    #we are going to look for other aircraft with
                         the exact same set of feasible starting lines
                    for it2 in active_feasibility_list:
                        options2 = active_feasibility_list[it2]
                        #a deadlock can also occur if there is 1 AC with not exactly the same
                             set of possible Lines, but if all the possible Lines of the AC
                             are in the potential deadlock. For example [2,14], [2,7,14] and
                        [2,7,14] is still a deadlock
                    if options2 == options1 or all(elements in options1 for elements in
                        options2):
                        number_of_duplicates += 1
                if number_of_duplicates == len(options1): #if the number of duplicates
                    equals the number of feasible starting lines then there is a deadlock
                    for it3 in options1:
                        if it3 not in deadlock_Lines:
                            deadlock_Lines.append(it3) #identify the line as a deadlock
                                line
                            k = 0
        for it in deadlock_Lines:
            possible_Lines.pop(it,None) #remove the deadlock lines from the possible lines,
                so that they wont get chosen

        #Now we have the list of Lines that can be selected for our problem we need to copy
            certain things to get it working
        possible_connections = {}
        possible_Lines_starting_each_day = {}
        possible_Lines_ending_each_day = {}
        possible_Lines_active_each_day = {}
        possible_Start_Lines = []
        for it in Connections:
            if it in possible_Lines:
                possible_connections[it]=[]
                for it2 in Connections[it]:
                    if it2 in possible_Lines:
                        possible_connections[it].append(it2)
        for it in Lines_starting_each_day:
            possible_Lines_starting_each_day[it]=[]
            for it2 in Lines_starting_each_day[it]:
                if it2 in possible_Lines:
                    possible_Lines_starting_each_day[it].append(it2)
        for it in Lines_ending_each_day:
            possible_Lines_ending_each_day[it]=[]
            for it2 in Lines_ending_each_day[it]:
                if it2 in possible_Lines:
                    possible_Lines_ending_each_day[it].append(it2)
        for it in Lines_active_each_day:
            possible_Lines_active_each_day[it]=[]
            for it2 in Lines_active_each_day[it]:
                if it2 in possible_Lines:
                    possible_Lines_active_each_day[it].append(it2)
        for it in OptionalStartLines[n]:
            if it in possible_Lines:
                possible_Start_Lines.append(it)

        My_X_Variables = tuplelist([])
        for i in aircraftID:
            for j in possible_Lines:
                My_X_Variables.append((i,j))
        My_Y_Variables = tuplelist([])
        for i in aircraftID:
            for t in Tasks[TailNumbers[i]]:
                for j in possible_Lines:
                    My_Y_Variables.append((i,t,j))
        #the following help variables are used to calculate the number of lost flying hours.
            This was necessary because a MIN or MAX function only works in a constraint and
            not in an objective function in Gurobi
        LFH = {}
        LFH2 = {}
        LFH3 = {}
        for i in aircraftID:
            for t in Tasks[TailNumbers[i]]:
                for j in possible_Lines:
                    LFH[i,t,j] = m.addVar(lb = -1000, ub = 1000)
                    LFH2[i,t,j] = m.addVar(lb = -1000, ub = 1000)
                    LFH3[i,t,j] = m.addVar(lb = -1000, ub = 1000)

        x = m.addVars(My_X_Variables, vtype=GRB.BINARY, name='x')
        y = m.addVars(My_Y_Variables, vtype=GRB.BINARY, name='y')   # equals 1 if task t is
```

```
    planned AFTER line j

m.setObjective(quicksum(y[i,t,j]*Tasks[TailNumbers[i]][t]['TaskMH']*LFH3[i,t,j] for i
     in aircraftID for j in possible_Lines for t in Tasks[TailNumbers[i]]),GRB.
    MINIMIZE)

#----- Routing Section

#The selected AC can fly at most one line per day. this also includes the first one:
for d in range(StartingDay,StartingDay+Planning_horizon):
    m.addConstrs(quicksum(x[i,j] for j in possible_Lines_active_each_day[d] ) <= 1
        for i in aircraftID)

#The first line of an AC should start from its initial location:
m.addConstrs(quicksum(x[i,j] for j in possible_Start_Lines) == 1 for i in aircraftID)

#If the selected AC flies line j, then it also must fly a connecting line, this
    excludes the last day
m.addConstrs(x[i,j] - quicksum(x[i,p] for p in possible_connections[j]) <= 0 for i in
     aircraftID for j in possible_connections)

#----- Task Planning

#Every task can at most be planned once:
m.addConstrs(quicksum(y[i,t,j] for j in possible_Lines) <= 1 for i in aircraftID for
    t in Tasks[TailNumbers[i]])

##All tasks in T_i which are due before a due date should be planned if that due date
    is closer than the planning horizon
m.addConstrs(-1000*quicksum(y[i,t,j]*MS[j] for j in possible_Lines) <= Tasks[
    TailNumbers[i]][t]['Task Due Date'] - (StartingDay+Planning_horizon+1) for i in
    aircraftID for t in Tasks[TailNumbers[i]])

#All tasks in T_i which are due before X flight hours should be planned if the
    selected route is longer than X flight hours.
m.addConstrs(-1000*quicksum(y[i,t,j]*MS[j] for j in possible_Lines) <= (Tasks[
    TailNumbers[i]][t]['Task FH Due'] - quicksum(x[i,j]*Lines[j]['Flight Hours'] for
    j in possible_Lines)) for i in aircraftID for t in Tasks[TailNumbers[i]])

#If task t of AC i is scheduled, it must be scheduled before the due date:
m.addConstrs(y[i,t,j]*Lines[j]['EndDate']+1 <= Tasks[TailNumbers[i]][t]['Task Due
    Date'] for i in aircraftID for t in Tasks[TailNumbers[i]] for j in possible_Lines
    )

#If task t of AC i is scheduled, it must be scheduled before the due flight hours:
m.addConstrs(y[i,t,j]*(Lines[j]['Flight Hours'] + quicksum(x[i,p]*Lines[p]['Flight
    Hours'] for d in range(StartingDay,Lines[j]['StartDate']) for p in
    possible_Lines_starting_each_day[d])) <= Tasks[TailNumbers[i]][t]['Task FH Due']
    for i in aircraftID for t in Tasks[TailNumbers[i]] for j in possible_Lines)

#A task t of AC i can only be after a line that is flown by AC i:
m.addConstrs(len(Tasks[TailNumbers[i]])*x[i,j] >= quicksum(y[i,t,j] for t in Tasks[
    TailNumbers[i]]) for i in aircraftID for j in possible_Lines)

#----- The capacity constraint
for d in range(StartingDay,StartingDay+Planning_horizon):
    m.addConstr(quicksum(y[i,t,j]*Tasks[TailNumbers[i]][t]['TaskMH'] for i in
        aircraftID for j in possible_Lines_ending_each_day[d] for t in Tasks[
        TailNumbers[i]]) <= active_Maint_capacity[d])

#Help variable for the objective function:
for i in aircraftID:
    for j in possible_Lines:
        for t in Tasks[TailNumbers[i]]:
            m.addConstr(LFH[i,t,j] == Tasks[TailNumbers[i]][t]['Task FH Due']- (Lines
                [j]['Flight Hours'] + quicksum(x[i,k]*Lines[k]['Flight Hours'] for d
                in range(StartingDay,Lines[j]['StartDate']) for k in
                possible_Lines_starting_each_day[d]))) #lost flying hours due to task
                 being executed before remaining legal FH
            m.addConstr(LFH2[i,t,j] == average_FH_per_day*(Tasks[TailNumbers[i]][t]['
                Task Due Date'] -(Lines[j]['EndDate']+1))) #Lost flying hours due to
```

```
                            task being executed before its due date
                        m.addConstr(LFH3[i,t,j] == min_(LFH[i,t,j],LFH2[i,t,j]))

        m.optimize()

        Line_solution = m.getAttr('x', x)
        task_solution = m.getAttr('x', y)
        Flying_Schedule[n] = [(i,j) for (i,j) in Line_solution if Line_solution[i,j] > 0.9]
        Maintenance_Schedule[n] = [(i,t,j) for (i,t,j) in task_solution if task_solution[i,t,
            j] > 0.9]
        objective_score[n] = m.objval

        #Now the network, constraints and lists need to be updated
        #remove the used Lines
        for it in Flying_Schedule[n]:
            active_Lines.pop(it[1])
        #update the remaining maintenance capacity
        MH_sum = {}
        quick_task_solution = []
        for it in Maintenance_Schedule[n]:
            quick_task_solution.append([it[0],it[1], Lines[it[2]]['EndDate']])
        for d in range(StartingDay,StartingDay+Planning_horizon):
            MH_sum[d]=0
        for it in quick_task_solution:
            MH_sum[it[2]] += Tasks[TailNumbers[it[0]]][it[1]]['TaskMH']
        for d in range(StartingDay,StartingDay+Planning_horizon):
            active_Maint_capacity[d] = active_Maint_capacity[d] - MH_sum[d]

        #update the feasibility list by removing used lines
        for it in Flying_Schedule[n]:
            for it2 in active_feasibility_list:
                if it[1] in active_feasibility_list[it2]:
                    active_feasibility_list[it2].remove(it[1])
    #now look at the combined scores of all the AC and save the score and planning if its the
        best
    Total_Score = sum(objective_score.values())
    ScoreTracker[iteration] = Total_Score
    ListTracker[iteration] = aircraft_list
    if Total_Score < Current_Solution_Score:
        Current_Solution_Score = Total_Score
        Current_Routing_Solution = Flying_Schedule
        Current_Planning_Solution = Maintenance_Schedule


#print the solutions
print('Number of AC: ' + str(len(TailNumbers)))
stopwatch_end = time.time()-stopwatch_start
print('Total Elapsed time: '+str(stopwatch_end))
print('Final Score :' +str(Current_Solution_Score))
```

D
Python Code - Successive Route Checking Matheuristic

```
# -*- coding: utf-8 -*-
"""
Created on Mon May  4 18:37:09 2020

@author: Alex
This script executes the successive route checking matheuristic
"""
import pandas as pd
from Line_extraction import Lines_Only
from gurobipy import *
import random
import time
import copy

TaskData = pd.read_excel('TaskData_20_tasks.xlsx', sheet_name=None)
FlightData = pd.read_excel('FlightData.xlsx', sheet_name=None)
TailNumbers = ['PH-BGF','PH-BGG','PH-BGH','PH-BGI','PH-BGK'] #this list contains all the tail
    numbers that are considered for the problem

Planning_horizon = 7
StartingDay = 43831 #Excel uses a number for a date, which works very well. 43831 translates
    to 1-1-2020
Station_Capacity = 150 #in man-hours per night

Maint_capacity = {}
for i in range(StartingDay,StartingDay+Planning_horizon):
    Maint_capacity[i] = Station_Capacity

#extract the lines from the flightdata. Within the function Lines_Only also a due date for
    the first standard check is randomly generated
Lines, StartLines, EndLines,standard_check_due_date = Lines_Only(FlightData,TailNumbers)
#calculate the average flight hours flown per day
average_FH_per_day = round(sum([Lines[b]['Flight Hours'] for b in Lines])/(len(TailNumbers)*
    Planning_horizon),1)

#%% determine all possible routes and their lengths
Connections = {}
for i in Lines:
    Connections[i] = []
    ThisLOF = Lines[i]
    EndDateThisLOF = ThisLOF['EndDate']
    for j in Lines:
        if Lines[j]['StartDate'] == EndDateThisLOF+1:
            Connections[i].append(j)
for i in EndLines:
    Connections.pop(i)

routes = {}
routeslength= {}
RouteCounter = 0
for i in StartLines:
    for i2 in Connections[i]:
        if i2 in EndLines:
            route = [i,i2]
            routes[RouteCounter] = route
            routeslength[RouteCounter] = sum([Lines[b]['Flight Hours'] for b in route])
            RouteCounter += 1
        else:
            for i3 in Connections[i2]:
                if i3 in EndLines:
                    route =[i,i2,i3]
                    routes[RouteCounter] = route
                    routeslength[RouteCounter] = sum([Lines[b]['Flight Hours'] for b in route
                        ])
                    RouteCounter += 1
                else:
                    for i4 in Connections[i3]:
                        if i4 in EndLines:
                            route =[i,i2,i3,i4]
                            routes[RouteCounter] = route
```

```
                                routeslength[RouteCounter] = sum([Lines[b]['Flight Hours'] for b
                                    in route])
                                RouteCounter += 1
                            else:
                                for i5 in Connections[i4]:
                                    if i5 in EndLines:
                                        route =[i,i2,i3,i4,i5]
                                        routes[RouteCounter] = route
                                        routeslength[RouteCounter] = sum([Lines[b]['Flight Hours'
                                            ] for b in route])
                                        RouteCounter += 1
                                    else:
                                        for i6 in Connections[i5]:
                                            if i6 in EndLines:
                                                route =[i,i2,i3,i4,i5,i6]
                                                routes[RouteCounter] = route
                                                routeslength[RouteCounter] = sum([Lines[b]['
                                                    Flight Hours'] for b in route])
                                                RouteCounter += 1
                                            else:
                                                for i7 in Connections[i6]:
                                                    if i7 in EndLines:
                                                        route =[i,i2,i3,i4,i5,i6,i7]
                                                        routes[RouteCounter] = route
                                                        routeslength[RouteCounter] = sum([Lines[b
                                                            ]['Flight Hours'] for b in route])
                                                        RouteCounter += 1

longest_route = max(routeslength, key=routeslength.get) #we have now found the longest
    possible route. This will determine which tasks are to considered

#In the given flight data Amsterdam is the maintenance station
#Whether or not Lines end on amsterdam. this is necessary for the ending lines, which could
    end on different stations than Amsterdam.
MS = {} #MS = maintenance station
for j in Lines:
    if Lines[j]['Arrival'] == 'Amsterdam':
        MS[j] = 1
    else:
        MS[j] = 0

#%% Determine the set of tasks for each aircraft that might need planning
ConsideredTaskData = {}
for i in TailNumbers:
    ConsideredTaskData[i] = pd.DataFrame(columns=('Task Number', 'Task MH', 'Task Due','FH
        Due'))
    counter = 0
    for index, row in TaskData[i].iterrows():
        DueDate = row['Task Due']
        DueFH = row['FH Due']
        if DueDate <= (StartingDay + Planning_horizon) and DueDate > StartingDay or DueFH <=
            routeslength[longest_route]:
            ConsideredTaskData[i].loc[counter] = [row['Task Number']] + [row['Task MH']] + [
                row['Task Due']] + [row['FH Due']]
            counter += 1
#sum of all considered tasks
Total_number_of_considered_tasks = sum([len(ConsideredTaskData[TN]) for TN in
    ConsideredTaskData])

#%% make list of all routes that can be traveled per tailsign
StartPoint = {}
#Determine the starting points for each tail
for i in range(len(StartLines)):
    #the first Startline is from the first aircraft and so on
    StartPoint[TailNumbers[i]] = Lines[i]['Departure']

#Out of all the possible routes created above, we now take a look at which routes are
    feasible for each tail number
#we also look at which startinglines are feasible for each tail number, because we need this
    to construct the feasibility list
Optional_routes = {}
```

```
OptionalStartLines = {}
for i in TailNumbers:
    Optional_routes[i] = []
    OptionalStartLines[i] = []
    #find earliest task due by date and by FH
    thisdue = StartingDay + Planning_horizon
    thisdueFH = 5000 #arbitrary large number
    for index, row in ConsideredTaskData[i].iterrows():
        if row['Task Due'] < thisdue:   #find the earliest task by due date for AC i
            thisdue = row['Task Due']
        if row['FH Due'] < thisdueFH:   #find the earlies task by flight hours for AC i
            thisdueFH = row['FH Due']

    for j in routes:
        if Lines[routes[j][0]]['Departure'] == StartPoint[i]:                    #Needs to
            start from correct station
            if Lines[routes[j][0]]['EndDate'] < standard_check_due_date[i]:      #Need to be
                able to perfrom first standard check
                if Lines[routes[j][0]]['EndDate'] < thisdue:                     #Needs to be
                     able to perform first task by due date
                    if Lines[routes[j][0]]['Flight Hours'] <= thisdueFH:          #Needs to
                        be able to perform first task by FH
                        Optional_routes[i].append(j)          #if all these conditions are
                            met, then the route is feasible for AC i
    for j in StartLines:
        if Lines[StartLines[j]]['Departure'] == StartPoint[i]:                    #Needs to
            start from correct station
            if Lines[StartLines[j]]['EndDate'] < standard_check_due_date[i]:      #Need to be
                 able to perfrom first standard check
                if Lines[StartLines[j]]['EndDate'] < thisdue:                     #Needs to
                    be able to perform first task by due date
                    if Lines[StartLines[j]]['Flight Hours'] <= thisdueFH:          #Needs to
                        be able to perform first task by FH
                        OptionalStartLines[i].append(j)


#%% Feasibility list
feasibility_list = copy.deepcopy(OptionalStartLines)
#if 1 AC has only 1 possible start LOF and 1 other AC has 2, but 1 of those is that singular
    possible startLOF for the other, than AC 2 also only has 1 possible start
k = 0
reserved_list=[]
while k == 0:
    k = 1
    for i in feasibility_list: #for every AC
        for p in feasibility_list[i]: #for every one of the feasible starting lines for the
            considered AC
            if p in reserved_list:  #if this starting line is reserved for another AC
                if len(feasibility_list[i]) > 1: #if there is more than 1 feasible start line
                     remaining
                    feasibility_list[i].remove(p)   #then remove it as a feasible line
                    k=0
        if len(feasibility_list[i]) == 1: #if there is only 1 feasible line left for the AC
            under consideration
            if feasibility_list[i][0] not in reserved_list: #and its not reserved by another
                AC
                reserved_list.append(feasibility_list[i][0])    ##then reserve it for this AC

#%% Begin the heuristic
number_of_iterations = 10
Current_Solution_Score = 100000 #arbitrary large number
i = int(time.time()*1000) #random seed based on current time, to always have random aircraft
    list
random.seed(i)
stopwatch_start = time.time()
Best_Score = 1000000 #arbitrary large number
iteration_counter = 0
seedtracker = {}
for iteration in range(number_of_iterations):
    Saved_Routes = {}
    Saved_Maint_Planning = {}
```

```
Saved_score = {}
#we create an aircraft list in a random order. In this order each aircraft will be
    assigned a route
aircraft_list = random.sample(TailNumbers,len(TailNumbers))
#Because within a decomposition by aircraft method, not all routes are created at the
    same time, we refer to the remaining routes (that have not been picked by a previous
    AC) as 'active'
active_routes = copy.deepcopy(Optional_routes)
active_Maint_capacity = copy.deepcopy(Maint_capacity) #the remaining maintenance capacity
active_feasibility_list = copy.deepcopy(feasibility_list)

for TN in aircraft_list: #start assigning routes to the aircraft one by one
    Score_per_route = {} #a dictionary to save the score for each route that will be
        checked
    Maint_Planning_per_route = {} #a dictionary to save the planning for each route that
        will be checked

    #------------------------------ Start Feasibility list Creation
        ------------------------------------

    active_feasibility_list.pop(TN)  #first remove current AC from feasibility list
    possible_routes = copy.deepcopy(active_routes[TN]) #we start off with all active
        lines as possible lines

    #the feasibility list will select a number of lines that cannot be chosen to
        guarantee feasible routes for upcoming aircraft
    ReservedLines = []
    routes_to_be_removed = []
    for i in active_feasibility_list:
        if len(active_feasibility_list[i]) <= 1:  #if there is only 1 feasible starting
            line remaining for a certain AC, then the current AC may not choose this line
            ReservedLines.append(active_feasibility_list[i][0])    #make a list of all
                the reserved Lines
    for i in possible_routes:         #for all our possible routes
        if routes[i][0] in ReservedLines:       #if the first line of the route is a
            reserved line for a different AC
            routes_to_be_removed.append(i)
    for i in routes_to_be_removed:
        possible_routes.remove(i)
    #Sometimes a 'deadlock' can occur: 2 AC both have the same 2 possible Lines, then
        these 2 Lines must be reserved for these AC, if another AC takes one of these the
         result will be infeasible
    #Thats why we have to spot and remove deadlocks:
    deadlock_Lines = []
    k = 0
    while k == 0:       #one deadlock being removed can lead to another deadlock
        appearing, thats why we have to iterate until no more deadlocks can be found
        k=1
        for it1 in active_feasibility_list: #for each tail number (TN) remaining in the
            feasibility list
            if len(active_feasibility_list[it1]) > 1:
                options1 = active_feasibility_list[it1] # a list of the feasible lines
                    for the TN
                number_of_duplicates = 0 #we are going to look for other aircraft with
                    the exact same set of feasible starting lines
                for it2 in active_feasibility_list:
                    options2 = active_feasibility_list[it2]
                    #a deadlock can also occur if there is 1 AC with not exactly the same
                         set of possible Lines, but if all the possible Lines of the AC
                        are in the potential deadlock. For example [2,14], [2,7,14] and
                        [2,7,14] is still a deadlock
                    if options2 == options1 or all(elements in options1 for elements in
                        options2):
                        number_of_duplicates += 1
                if number_of_duplicates == len(options1): #if the number of duplicates
                    equals the number of feasible starting lines then there is a deadlock
                    for it3 in options1:
                        if it3 not in deadlock_Lines:
                            deadlock_Lines.append(it3) #identify the line as a deadlock
                                line
                            k = 0
    routes_to_be_removed = []
```

```
        for it in possible_routes:    #The deadlock Lines have been identified , the
            corresponding routes must be removed
            if routes[it][0] in deadlock_Lines:
                routes_to_be_removed.append(it)
        for it in routes_to_be_removed:
            possible_routes.remove(it)
        #----------------------------- End Feasibility list Creation
            ------------------------------------

        #Now that the set of possible routes has been created we need to find the best one
        for i in possible_routes:
            iteration_counter += 1
            #Determine the Lines that are in our route and their recorded flight hours at the
                 end of each line
            flownFHbeforelineends = {}
            previouslyflownlinesfh = 0
            for it in routes[i]:
                flownFHthisline = Lines[it]['Flight Hours']
                flownFHbeforelineends[it] = flownFHthisline + previouslyflownlinesfh
                previouslyflownlinesfh += flownFHthisline

            #now that we have selected the route and the amount of FH are known we can create
                 a truth-based task list, containing all tasks that should be planned within
                the planning horizon
            Tasks = {}
            FH_this_route = routeslength[i] #we previously determined how long each route was
                 in flight-hours
            for index, row in ConsideredTaskData[TN].iterrows():
                #Should the task be considered in this time horizon? --> check the amount of
                    FH in the route
                DueDate = row['Task Due']
                DueFH = row['FH Due']
                if DueDate <= (StartingDay + Planning_horizon) and DueDate > StartingDay or
                    DueFH <= FH_this_route:
                    Task = {'Task Number':int(row['Task Number']),
                            'TaskMH': row['Task MH'],
                            'Task Due Date' : row['Task Due'],
                            'Task FH Due' : row['FH Due']
                        }
                    Tasks[int(row['Task Number'])] = Task  #The int is because the task
                        number were given as 3.0 and i wanted them as 3

            m = Model("Successive_route_checking_matheuristic")

            My_Y_Variables = tuplelist([])
            for j in routes[i]:
                for k in Tasks:
                    My_Y_Variables.append((j,k))

            y = m.addVars(My_Y_Variables, vtype=GRB.BINARY, name='x') #equals 1 if task t is
                planned after line j
            #minimize the lost flying hours
            m.setObjective(quicksum(y[j,k]*Tasks[k]['TaskMH']*min(average_FH_per_day*(Tasks[k
                ]['Task Due Date']-(Lines[j]['EndDate']+1)),(Tasks[k]['Task FH Due']-
                flownFHbeforelineends[j])) for j in routes[i] for k in Tasks),GRB.MINIMIZE)

            #Every task must be planned on a maint station
            m.addConstrs(quicksum(y[j,k]*MS[j] for j in routes[i]) == 1 for k in Tasks)

            #Tasks must be performed before their due date
            m.addConstrs((y[j,k]*Tasks[k]['Task Due Date'] >= y[j,k]*(Lines[j]['EndDate']+1)
                for k in Tasks for j in routes[i]),"constraint4")

            #Tasks must be performed before their due FH
            m.addConstrs((y[j,k]*Tasks[k]['Task FH Due'] >= y[j,k]*flownFHbeforelineends[j]
                for k in Tasks for j in routes[i]),"constraint4")

            #tasks have to fit inside the groundtime
            m.addConstrs(quicksum(y[j,k]*Tasks[k]['TaskMH'] for k in Tasks) <= Maint_capacity
                [Lines[j]['EndDate']] for j in routes[i] )
            m.optimize()

            Planning_solution = m.getAttr('x',y)
            nonzero_planning_solution = [p for p in Planning_solution if Planning_solution[p]
                 > 0.9]
            Score_per_route[i] = m.objVal #we save the score for each route, to be able to
                select the best one eventually
            Maint_Planning_per_route[i] = nonzero_planning_solution #we save the maintenance
                planning corresponding with each route, to be able to select the best one
                eventually

        #Get the route corresponding to the best solution:
        selected_route = min(Score_per_route, key=Score_per_route.get)
        Saved_score[TN] = Score_per_route[selected_route]
        Saved_Routes[TN] = selected_route
        Saved_Maint_Planning[TN] = Maint_Planning_per_route[selected_route]
        #Pop the routes that now can no longer be flown because of the used Lines
        UsedLines = routes[selected_route]
        PoppedRoutes = []
        for p in TailNumbers:
            for u in active_routes[p]:
                for o in routes[u]:
                    if o in UsedLines and u not in PoppedRoutes:
                        PoppedRoutes.extend([u])
        for p in TailNumbers:
            for x in PoppedRoutes:
                if x in active_routes[p]:
                    active_routes[p].remove(x)

        #update the feasibility list by removing used lines
        for it in UsedLines:
            for it2 in active_feasibility_list:
                if it in active_feasibility_list[it2]:
                    active_feasibility_list[it2].remove(it)
        #update the remaining maintenance capacity per day
        for j,t in Saved_Maint_Planning[TN]: #j is line, t is task
            theday = Lines[j]['EndDate'] #at which day was it planned
            usedMH = float(ConsideredTaskData[TN].loc[ConsideredTaskData[TN]['Task Number']
                == t]['Task MH']) #locate the task (it[0]) within the task dataframe and
                select its size in man-hours
            active_Maint_capacity[theday] = active_Maint_capacity[theday] - usedMH #remove
                the used man-hours of the task from the remaining maintenance capacity

    #We save the best result that comes with each iteration
    Iteration_Score = sum(Saved_score.values())
    Iteration_Routing = Saved_Routes
    Iteration_Planning = Saved_Maint_Planning
    if Iteration_Score < Best_Score: #if the iteration score is better than the best score so
         far
        Best_Score = Iteration_Score #save the iteration score as the new best score
        Best_Routing = copy.deepcopy(Iteration_Routing) #save its routing
        Best_Planning = copy.deepcopy(Iteration_Planning) #save its maintenance schedule

print('Number of AC: ' + str(len(TailNumbers)))
stopwatch_end = time.time()-stopwatch_start
print('Total Elapsed time: '+str(stopwatch_end))
print('The best found score: ' + str(Best_Score))
```

E
Python Code - Rolling Horizon Matheuristic

```
# -*- coding: utf-8 -*-
"""
Created on Tue May 12 10:19:54 2020

@author: Alex
In this script we execute the rolling horizon matheuristic without forecasting
"""

import pandas as pd
from Line_extraction import Lines_Only #this is a function that will extract lines from
    flightdatas
from gurobipy import *
import copy
import time

TaskData = pd.read_excel('TaskData_20_tasks.xlsx', sheet_name=None)
FlightData = pd.read_excel('FlightData.xlsx', sheet_name=None)
TailNumbers = pd.ExcelFile('FlightData.xlsx').sheet_names
TailNumbers = ['PH-BGF','PH-BGG','PH-BGH','PH-BGI','PH-BGK'] #this list contains all the tail
     numbers that are considered for the problem
#Parameters
Planning_horizon = 7
StartingDay = 43831 #Excel uses a number for a date, which works very well. 43831 translates
    to 1-1-2020
Station_Capacity = 150 #in man-hours per night

#extract the lines from the flightdata. Within the function Lines_Only also a due date for
    the first standard check is randomly generated
Lines, StartLines, EndLines,standard_check_due_date = Lines_Only(FlightData,TailNumbers)
#calculate the average flight hours flown per day
average_FH_per_day = round(sum([Lines[b]['Flight Hours'] for b in Lines])/(len(TailNumbers)*
    Planning_horizon),1)

#I make a dictionary here containing all the task data to which i can always refer in the
    code
AllTaskData = {}
for i in TailNumbers:
    AllTaskData[i] = {}
    for index, row in TaskData[i].iterrows():
        task_number = int(row['Task Number'])
        Task = {'Task Number':int(row['Task Number']),
            'TaskMH': row['Task MH'],
            'Task Due Date' : row['Task Due'],
            'Task FH Due' : row['FH Due']
        }
        AllTaskData[i][task_number] = Task

#%% Preprocessing for building the routes
#Here multiple dictionaries are built containing line information, which are used when
    defining the model
Lines_starting_each_day = {}
Lines_ending_each_day = {}
for i in range(Planning_horizon):
    for j in Lines:
       Lines_starting_each_day[StartingDay+i] = [a for a in Lines if Lines[a]['StartDate'] ==
            StartingDay+i]
       Lines_ending_each_day[StartingDay+i] = [a for a in Lines if Lines[a]['EndDate'] ==
           StartingDay+i]

#In the given flight data Amsterdam is the maintenance station
#Whether or not Lines end on amsterdam. this is necessary for the ending lines, which could
    end on different stations than Amsterdam.
MS = {}  #MS = maintenance station
for j in Lines:
    if Lines[j]['Arrival'] == 'Amsterdam':
        MS[j] = 1
    else:
        MS[j] = 0

#Determine the starting points for each tail
```

```
StartPoint = {}
for i in range(len(TailNumbers)):
    Starter = StartLines[i]
    #the first Startline is from the first aircraft and so on
    StartPoint[TailNumbers[i]] = Lines[Starter]['Departure']
#Now from the startpoint determine the set of 1st day Lines possible for each tailsign
PossibleStartLines = {}
for i in TailNumbers:
    PossibleStartLines[i] = [j for j in StartLines if Lines[j]['Departure'] == StartPoint[i]
        and Lines[j]['EndDate'] < standard_check_due_date[i]]

#%% Start the matheuristics
#The encountered lines are divided in frozen lines and active lines. frozen lines have been
    assigned to aircraft already. active lines will be assigned during this iteration
#there are also active aircraft and non-active aircraft. active aircraft will be assigned an
    active line during the iterations.
#both active and non-active AC have to replan their maintenance every iteration, to make sure
     the result is optimal.
#non-active AC are replanning maintenance tasks they have already planned before, but because
     of the capacity constraint, they must plan them again simulateniously with the active AC
active_aircraft = copy.deepcopy(TailNumbers)
non_active_aircraft = []
all_aircraft = active_aircraft + non_active_aircraft
non_active_task_set = {}
Flight_Hours_Recorded = {}
frozen_lines = []
Constructed_Routes = {}
Most_Recent_Maint_Schedule = {}
Flown_FH_after_line = {}
Encountered_lines_ending_each_day = {}
HAFL = {}   #a dictionary that holds information on which frozen line has been assigned to
    which AC. HAFL: Has AC flown line?
MaintScore = {}
for i in TailNumbers:
    Flight_Hours_Recorded[i] = 0 #we need to keep track of the number of recorded flight
        hours before each iteration
    Constructed_Routes[i] = [] #we need to keep track of the constructed routes so far
    Most_Recent_Maint_Schedule[i] = []  #we need to keep track of the most recent maintenance
         schedule
    MaintScore[i] = 0
    HAFL[i] = {}
    for j in Lines:
        HAFL[i][j] = 0  #If AC i has flown line j, we need to keep track of this to know when
             an AC can plan its maintenance

stopwatch_start = time.time() #start a timer

for day in range(StartingDay,StartingDay+Planning_horizon):

    #Select the lines starting at this day (= active lines)
    active_lines = [b for b in Lines if Lines[b]['StartDate'] == day]
    if len(active_lines) > 0: #if there are active lines
        longest_line_FH = max([Lines[b]['Flight Hours'] for b in active_lines]) #what is the
            longest lines in FH?
        line_latest_enddate = max([Lines[b]['EndDate'] for b in active_lines]) #What is the
            latest end date of a line?

    Encountered_lines = frozen_lines + active_lines #maintenance can be planned both after
        frozen lines and after active lines
    #----- we want dictionaries containing the lines that end on each specific day for both
        active and non-active AC
    for d in range(StartingDay,StartingDay+Planning_horizon):
        Encountered_lines_ending_each_day[d] = [j for j in Lines_ending_each_day[d] if j in
            Encountered_lines]

    #this will be used in the capacity constraint for the non-active aircraft
    Frozen_line_endings_for_non_active_AC = {}
    for i in non_active_aircraft:
        Frozen_line_endings_for_non_active_AC[i] = {}
        for d in range(StartingDay,StartingDay+Planning_horizon):
            Frozen_line_endings_for_non_active_AC[i][d] = [j for j in Constructed_Routes[i]
```

```
                if Lines[j]['EndDate'] == d]
    #-------- Line ending dictionaries are now constructed

    #select the tasks that should be considered
    #to do this we need to acknowledge the flying history and the max possible length of the
        new lines
    Tasks = {}
    for TS in active_aircraft:
        Tasks[TS] = {}
        for index, row in TaskData[TS].iterrows():
            DueDate = row['Task Due']
            DueFH = row['FH Due']
            if DueDate <= (line_latest_enddate+1) or (DueFH - Flight_Hours_Recorded[TS]) <=
                longest_line_FH:
                task_number = int(row['Task Number'])
                Task = {'Task Number':task_number,
                    'TaskMH': row['Task MH'],
                    'Task Due Date' : row['Task Due'],
                    'Task FH Due' : row['FH Due']
                }
                Tasks[TS][task_number] = Task

    m = Model('RHM')

    My_X_Variables = tuplelist([])
    for i in range(len(active_aircraft)):
        for j in active_lines: #only active lines need to be distributed, frozen lines have
            already been distributed in a previous iteration
            My_X_Variables.append((i,j))
    My_Y_Variables = tuplelist([])
    for i in range(len(active_aircraft)):
        for t in Tasks[active_aircraft[i]]:
            for j in Encountered_lines: #an AC can plan maintenance both after an active
                line and after a frozen line
                My_Y_Variables.append((i,t,j))

    #the following help variables are used to calculate the number of lost flying hours. This
         was necessary because a MIN or MAX function only works in a constraint and not in an
         objective function in Gurobi
    LFH1_ac = {}
    LFH2_ac = {}
    LFH3_ac = {}
    LFH1_fr = {}
    LFH2_fr = {}
    LFH3_fr = {}
    for i in range(len(active_aircraft)):
        for t in Tasks[active_aircraft[i]]:
            for j in active_lines: #help variables are broken up in active and in frozen
                variants, because they need different formulations
                LFH1_ac[i,t,j] = m.addVar(lb = -10000, ub = 10000)
                LFH2_ac[i,t,j] = m.addVar(lb = -10000, ub = 10000)
                LFH3_ac[i,t,j] = m.addVar(lb = -10000, ub = 10000)
            for j in frozen_lines:
                LFH1_fr[i,t,j] = m.addVar(lb = -10000, ub = 10000)
                LFH2_fr[i,t,j] = m.addVar(lb = -10000, ub = 10000)
                LFH3_fr[i,t,j] = m.addVar(lb = -10000, ub = 10000)
    #this is for the non_active aircraft
    for i in range(len(active_aircraft),len(all_aircraft)):
        for t in non_active_task_set[all_aircraft[i]]: #the set of tasks that non-active AC
            have to schedule has been determined in a previous iteration
            for j in Constructed_Routes[all_aircraft[i]]: #non-active AC i can only plan
                after lines that it has flown
                My_Y_Variables.append((i,t,j))
                LFH1_fr[i,t,j] = m.addVar(lb = -10000, ub = 10000)
                LFH2_fr[i,t,j] = m.addVar(lb = -10000, ub = 10000)
                LFH3_fr[i,t,j] = m.addVar(lb = -10000, ub = 10000)

    x = m.addVars(My_X_Variables, vtype=GRB.BINARY, name='x')  # if line j is assigned to AC
         i
    y = m.addVars(My_Y_Variables, vtype=GRB.BINARY, name='y')  # equals 1 if AC i plans task
         t AFTER line j
    #The score calculation of planning after frozen and active lines is broken up, because
        they require different formulations. Also the non-active AC are seperate in the
        objective function
    m.setObjective(quicksum(y[i,t,j]*Tasks[active_aircraft[i]][t]['TaskMH']*LFH3_ac[i,t,j]
        for i in range(len(active_aircraft)) for j in active_lines for t in Tasks[
        active_aircraft[i]]) + quicksum(y[i,t,j]*Tasks[active_aircraft[i]][t]['TaskMH']*
        LFH3_fr[i,t,j] for i in range(len(active_aircraft)) for j in frozen_lines for t in
        Tasks[active_aircraft[i]]) + quicksum(y[i,t,j]*AllTaskData[all_aircraft[i]][t]['
        TaskMH']*LFH3_fr[i,t,j] for i in range(len(active_aircraft),len(all_aircraft)) for j
        in Constructed_Routes[all_aircraft[i]] for t in non_active_task_set[all_aircraft[i]])
         , GRB.MINIMIZE)

#----- Routing Section
    #every active line must be flown
    m.addConstrs(quicksum(x[i,j] for i in range(len(active_aircraft))) == 1 for j in
        active_lines)
    #Every active AC must fly exactly one active line
    m.addConstrs(quicksum(x[i,j] for j in active_lines) == 1 for i in range(len(
        active_aircraft)))
    #for the first day we can only choose a feasible starting line
    if day == StartingDay:
        m.addConstrs(quicksum(x[i,j] for j in PossibleStartLines[active_aircraft[i]]) == 1
            for i in range(len(active_aircraft)))

#----- Task Planning
    #Every considered task can be planned at most once, after an active line or a frozen line
    m.addConstrs(quicksum(y[i,t,j] for j in Encountered_lines) <= 1 for i in range(len(
        active_aircraft)) for t in Tasks[active_aircraft[i]])
    #for non-active AC: all their tasks must be planned
    m.addConstrs(quicksum(y[i,t,j] for j in Constructed_Routes[all_aircraft[i]]) == 1 for i
        in range(len(active_aircraft),len(all_aircraft)) for t in non_active_task_set[
        all_aircraft[i]])

    ##All tasks in T_i which are due before a due date should be planned if that due date is
        closer than the end of the selected line
    m.addConstrs(-1000*quicksum(y[i,t,j]*MS[j] for j in Encountered_lines) <= Tasks[
        active_aircraft[i]][t]['Task Due Date'] - (quicksum(x[i,j]*Lines[j]['EndDate'] for j
        in active_lines)+2) for i in range(len(active_aircraft)) for t in Tasks[
        active_aircraft[i]])

    #All tasks in T_i which are due before X flight hours should be planned if the selected
        route is longer than X flight hours.
    m.addConstrs(-1000*quicksum(y[i,t,j]*MS[j] for j in Encountered_lines) <= (Tasks[
        active_aircraft[i]][t]['Task FH Due'] - (Flight_Hours_Recorded[active_aircraft[i]] +
        quicksum(x[i,j]*Lines[j]['Flight Hours'] for j in active_lines))) for i in range(len(
        active_aircraft)) for t in Tasks[active_aircraft[i]])

    #If task t of AC i is scheduled, it must be scheduled before the due date:
    m.addConstrs(y[i,t,j]*Lines[j]['EndDate']+1 <= Tasks[active_aircraft[i]][t]['Task Due
        Date'] for i in range(len(active_aircraft)) for t in Tasks[active_aircraft[i]] for j
        in Encountered_lines)
    #For non active AC as well:
    m.addConstrs(y[i,t,j]*Lines[j]['EndDate']+1 <= AllTaskData[all_aircraft[i]][t]['Task Due
        Date'] for i in range(len(active_aircraft),len(all_aircraft)) for t in
        non_active_task_set[all_aircraft[i]] for j in Constructed_Routes[all_aircraft[i]])

    #If task t of AC i is scheduled, it must be scheduled before the due flight hours:
    #This constraint is broken up in the frozen lines and the active lines, as they require
        different formulations
    for i in range(len(active_aircraft)):
        for t in Tasks[active_aircraft[i]]:
            #frozen lines:
            m.addConstrs(y[i,t,j]*Flown_FH_after_line[j]  <= Tasks[active_aircraft[i]][t]['
                Task FH Due'] for j in frozen_lines)
            #active lines:
            m.addConstrs(y[i,t,j]*(Flight_Hours_Recorded[active_aircraft[i]] + Lines[j]['
                Flight Hours']) <= Tasks[active_aircraft[i]][t]['Task FH Due'] for j in
                active_lines)
    #for non-active AC as well:
    m.addConstrs(y[i,t,j]*Flown_FH_after_line[j]  <= AllTaskData[all_aircraft[i]][t]['Task FH
        Due'] for i in range(len(active_aircraft),len(all_aircraft)) for t in
        non_active_task_set[all_aircraft[i]] for j in Constructed_Routes[all_aircraft[i]])

    #Only plan a task after a line if the AC has flown that line
    #This constraint is broken up in the frozen lines and the active lines, as they require
        different formulations
    for i in range(len(active_aircraft)):
       #frozen lines
        m.addConstrs(len(Tasks[active_aircraft[i]])*HAFL[active_aircraft[i]][j] >= quicksum(y
            [i,t,j] for t in Tasks[active_aircraft[i]]) for j in frozen_lines)
        #active lines
        m.addConstrs(len(Tasks[active_aircraft[i]])*x[i,j] >= quicksum(y[i,t,j] for t in
            Tasks[active_aircraft[i]]) for j in active_lines)

    #Capacity constraint:
    for d in range(StartingDay,StartingDay+Planning_horizon):
#        m.addConstr(quicksum(y[i,t,j]*Tasks[active_aircraft[i]][t]['TaskMH'] for i in range(
    len(active_aircraft)) for j in Encountered_lines_ending_each_day[d] for t in Tasks[
    active_aircraft[i]]) <= Station_Capacity)
        m.addConstr(quicksum(y[i,t,j]*Tasks[active_aircraft[i]][t]['TaskMH'] for i in range(
            len(active_aircraft)) for j in Encountered_lines_ending_each_day[d] for t in
            Tasks[active_aircraft[i]]) + quicksum(y[i,t,j]*AllTaskData[all_aircraft[i]][t]['
            TaskMH'] for i in range(len(active_aircraft),len(all_aircraft)) for j in
            Frozen_line_endings_for_non_active_AC[all_aircraft[i]][d] for t in
            non_active_task_set[all_aircraft[i]]) <= Station_Capacity)

    #Calculate the lost flying hours to help the objective function
    #This constraint is broken up in the frozen lines and the active lines, as they require
        different formulations
    for i in range(len(active_aircraft)):
        for j in active_lines: #for the active lines
            for t in Tasks[active_aircraft[i]]:
                m.addConstr(LFH1_ac[i,t,j] == Tasks[active_aircraft[i]][t]['Task FH Due']- (
                    Flight_Hours_Recorded[active_aircraft[i]] + Lines[j]['Flight Hours'])) #
                    lost flying hours due to task being executed before remaining legal FH
                m.addConstr(LFH2_ac[i,t,j] == average_FH_per_day*(Tasks[active_aircraft[i]][t
                    ]['Task Due Date'] -(Lines[j]['EndDate']+1))) #Lost flying hours due to
                    task being executed before its due date
                m.addConstr(LFH3_ac[i,t,j] == min_(LFH1_ac[i,t,j],LFH2_ac[i,t,j]))
        for j in frozen_lines: #for the frozen lines
            for t in Tasks[active_aircraft[i]]:
                m.addConstr(LFH1_fr[i,t,j] == Tasks[active_aircraft[i]][t]['Task FH Due'] -
                    Flown_FH_after_line[j]) #lost flying hours due to task being executed
                    before remaining legal FH
                m.addConstr(LFH2_fr[i,t,j] == average_FH_per_day*(Tasks[active_aircraft[i]][t
                    ]['Task Due Date'] -(Lines[j]['EndDate']+1))) #Lost flying hours due to
                    task being executed before its due date
                m.addConstr(LFH3_fr[i,t,j] == min_(LFH1_fr[i,t,j],LFH2_fr[i,t,j]))
        #for non-active aircraft as well
    for i in range(len(active_aircraft),len(all_aircraft)):
        for j in Constructed_Routes[all_aircraft[i]]:
            for t in non_active_task_set[all_aircraft[i]]:
                m.addConstr(LFH1_fr[i,t,j] == AllTaskData[all_aircraft[i]][t]['Task FH Due']
                    - Flown_FH_after_line[j]) #lost flying hours due to task being executed
                    before remaining legal FH
                m.addConstr(LFH2_fr[i,t,j] == average_FH_per_day*(AllTaskData[all_aircraft[i
                    ]][t]['Task Due Date'] -(Lines[j]['EndDate']+1))) #Lost flying hours due
                    to task being executed before its due date
                m.addConstr(LFH3_fr[i,t,j] == min_(LFH1_fr[i,t,j],LFH2_fr[i,t,j]))

    m.optimize()
    #Save routing solution and prepare for next day iteration
    line_solution = m.getAttr('x', x)
    Routing_Solution = [(i,j) for (i,j) in line_solution if line_solution[i,j] > 0.9]
    task_solution = m.getAttr('x', y)
    Maintenance_Planning_Solution = [(i,t,j) for (i,t,j) in task_solution if task_solution[i,
        t,j] > 0.9]

    #i need to update: frozen_lines, Flight_hours_recorded, constructed_route, HAFL and
        active_aircraft list
    for i,j in Routing_Solution:
        frozen_lines.append(j) #update the frozen lines
        HAFL[active_aircraft[i]][j] = 1 #update HAFL
        Constructed_Routes[active_aircraft[i]].append(j) #update Constructed Routes
        Flight_Hours_Recorded[active_aircraft[i]] += Lines[j]['Flight Hours'] #update the
            recorded flight hours
    #now reset the most recent maintenance schedule because we have replanned
    for i in range(len(all_aircraft)):
        Most_Recent_Maint_Schedule[all_aircraft[i]] = []
    #reconstruct the most recent maintenance schedule
    for i,t,j in Maintenance_Planning_Solution:
        Most_Recent_Maint_Schedule[all_aircraft[i]].append([t,j])

    #Now I need to specify how many flight hours were flown after each line in the frozen
        lines, for future planning needs
    for i in TailNumbers: #for every AC
        for m in range(len(Constructed_Routes[i])): #for every line that it has been assigned
             so far
            the_line = Constructed_Routes[i][m] #identify the line
            flown_lines_before_it = Constructed_Routes[i][:(m+1)] #identify which lines are
                assigned to the AC before the considered line
            Flown_FH_after_line[the_line] = sum([Lines[b]['Flight Hours'] for b in
                flown_lines_before_it]) #determine how many FH are flown before the line
                start

    #Keep track of the current scores of the routes
    for i in all_aircraft:
        MaintScore[i]=0 #reset the score for the active AC cause we are going to recalculate
            them
        for t,j in Most_Recent_Maint_Schedule[i]: #iterate through the planned maintenance
            for AC i
            Score_days = (AllTaskData[i][t]['Task Due Date'] - (Lines[j]['EndDate']+1))*
                average_FH_per_day #Lost flying hours due to task being executed before its
                due date
            Score_FH = AllTaskData[i][t]['Task FH Due'] - Flown_FH_after_line[j] #lost flying
                 hours due to task being executed before remaining legal FH
            overall_score = AllTaskData[i][t]['TaskMH']*min(Score_days,Score_FH)
            MaintScore[i] += overall_score

    #Now to determine the active aircraft for the upcoming iteration
    active_aircraft = []
    for i in TailNumbers: #for every AC
        Line_end = Lines[Constructed_Routes[i][-1]]['EndDate'] #determine when its most
            recently assigned line will end
        if Line_end == day: #if its line will end tonight
            active_aircraft.append(i) #then list it as an active AC
    non_active_aircraft = [] #the aircraft that are not active next iteration will still need
         to plan their maintenance again
    for i in TailNumbers:
        if i not in active_aircraft:
            non_active_aircraft.append(i)
    all_aircraft = active_aircraft + non_active_aircraft
    #determine the tasks that the non-active AC have planned and will need to replan next
        iteration
    for i in non_active_aircraft:
        non_active_task_set[i] = [i for (i,j) in Most_Recent_Maint_Schedule[i]]

#print solutions:
print('Final Scores: '+str(MaintScore))
print('Final Total Score: '+str(sum(MaintScore.values())))
stopwatch_end = time.time()-stopwatch_start
print('Total Elapsed time: '+str(stopwatch_end))
print('Total number of AC: ' +str(len(TailNumbers)))
```

F

Python Code - Rolling Horizon Matheuristic with Exact Forecasting

```python
# -*- coding: utf-8 -*-
"""
Created on Tue May 12 10:19:54 2020

@author: Alex
in this script we solve the RHM with exact forecasting. The number of days we forecast for is
     an input parameter.
"""

import pandas as pd
from Line_extraction import Lines_Only #this is a function that will extract lines from
    flightdata
from gurobipy import *
import copy
import time

TaskData = pd.read_excel('TaskData_20_tasks.xlsx', sheet_name=None)
FlightData = pd.read_excel('FlightData.xlsx', sheet_name=None)
TailNumbers = pd.ExcelFile('FlightData.xlsx').sheet_names
TailNumbers = ['PH-BGF','PH-BGG','PH-BGH','PH-BGI','PH-BGK'] #this list contains all the tail
     numbers that are considered for the problem

#Parameters
Planning_horizon = 7
StartingDay = 43831 #Excel uses a number for a date, which works very well. 43831 translates
    to 1-1-2020
Station_Capacity = 150 #in man-hours per night
solving_period = 3 #number of days for which to exactly solve the problem. If solving period
    = 3, we have a 2 day forecast

#extract the lines from the flightdata. Within the function Lines_Only also a due date for
    the first standard check is randomly generated
Lines, StartLines, EndLines,standard_check_due_date = Lines_Only(FlightData,TailNumbers)
#calculate the average flight hours flown per day
average_FH_per_day = round(sum([Lines[b]['Flight Hours'] for b in Lines])/(len(TailNumbers)*
    Planning_horizon),1)

#I make a dictionary here containing all the task data to which i can always refer in the
    code
AllTaskData = {}
for i in TailNumbers:
    AllTaskData[i] = {}
    for index, row in TaskData[i].iterrows():
        task_number = int(row['Task Number'])
        Task = {'Task Number':task_number,
            'TaskMH': row['Task MH'],
            'Task Due Date' : row['Task Due'],
            'Task FH Due' : row['FH Due']
        }
        AllTaskData[i][task_number] = Task

#%% Preprocessing for building the routes
#Here multiple dictionaries are built containing line information, which are used when
    defining the model
Lines_starting_each_day = {}
Lines_ending_each_day = {}
for i in range(Planning_horizon):
    for j in Lines:
       Lines_starting_each_day[StartingDay+i] = [a for a in Lines if Lines[a]['StartDate'] ==
            StartingDay+i]
       Lines_ending_each_day[StartingDay+i] = [a for a in Lines if Lines[a]['EndDate'] ==
           StartingDay+i]

#In the given flight data Amsterdam is the maintenance station
#Whether or not Lines end on amsterdam. this is necessary for the ending lines, which could
    end on different stations than Amsterdam.
MS = {} #MS: Maintenance station
for j in Lines:
    if Lines[j]['Arrival'] == 'Amsterdam':
        MS[j] = 1
    else:
        MS[j] = 0

#Determine the starting points for each tail
StartPoint = {}
for i in range(len(TailNumbers)):
    Starter = StartLines[i]
    #the first Startline is from the first aircraft and so on
    StartPoint[TailNumbers[i]] = Lines[Starter]['Departure']
#Now from the startpoint determine the set of 1st day Lines possible for each tailsign
PossibleStartLines = {}
for i in TailNumbers:
    PossibleStartLines[i] = [j for j in StartLines if Lines[j]['Departure'] == StartPoint[i]
        and Lines[j]['EndDate'] < standard_check_due_date[i]]

#%% Algorithm Start
#The encountered lines are divided in frozen lines and active lines. frozen lines have been
    assigned to aircraft already. active lines will be assigned during this iteration
#there are 3 types of aircraft: 1. starting aircraft start from the first day of the central
    period and their lines will be frozen
#2. transit AC will start a new line during the forecasting period but not during the central
     period
#3. non-active AC will not start a new line during either the central or forecasting period
starting_aircraft = copy.deepcopy(TailNumbers)
transit_aircraft = []
active_aircraft = starting_aircraft + transit_aircraft #active AC are all the AC that will be
     assigned lines during the iteration (not all lines will be frozen)
non_active_aircraft = []
all_aircraft = active_aircraft + non_active_aircraft
non_active_task_set = {}
Flight_Hours_Recorded = {} #a dict that will save the number of FH each AC has flown
    throughout the iterations
frozen_lines = [] #a list of all lines that have been assigned and frozen
Constructed_Routes = {} #in this dict we will save the frozen routes
Most_Recent_Maint_Schedule = {} #a dict that holds the maintenance schedule, which is updated
     during every iteration
Flown_FH_after_line = {} #this dict will save the number of FH that are recorded after
    completing a frozen line
Encountered_lines_ending_each_day = {}
HAFL = {}   #a dictionary that holds information on which frozen line has been assigned to
    which AC. HAFL: Has AC flown line?
MaintScore = {} #a dict that will save the scores of the maintenance planning for each AC
for i in TailNumbers:
    Flight_Hours_Recorded[i] = 0
    Constructed_Routes[i] = []
    Most_Recent_Maint_Schedule[i] = []
    MaintScore[i] = 0
    HAFL[i] = {}
    for j in Lines:
        HAFL[i][j] = 0

stopwatch_start = time.time() #start a timer

for day in range(StartingDay,StartingDay+Planning_horizon):

    #Select the lines starting at this day
    active_lines = []
    active_lines_starting_each_day = {}
    starting_active_lines = [] #list of all active lines that start on the first day
    future_lines = [] #list of lines that will start within the current forecasting period
    for i in range(day,day+solving_period):
        active_lines_starting_each_day[i] = [b for b in Lines if Lines[b]['StartDate'] == i]
        active_lines.extend([b for b in Lines if Lines[b]['StartDate'] == i])
        if i == day:
            starting_active_lines.extend([b for b in Lines if Lines[b]['StartDate'] == i])
        else:
            future_lines.extend([b for b in Lines if Lines[b]['StartDate'] == i])
    #determine the lines that are the last lines if they are distributed
    ending_lines = []
    for i in active_lines:
        if Lines[i]['EndDate']+1 > day+solving_period-1 or Lines[i]['EndDate']+1 >=
            StartingDay+Planning_horizon: #a line is an ending line if the morning at which
            the next line will start falls outside of the forecasting horizon OR if it falls
            outside of the total planning horizon (at the end)
            ending_lines.append(i)
    #to succesfully define the routing i use 2 connections dictionaries. One with active
        lines connections and one with already assigned line connections
    active_connections = {} #a dict that will contain connections between active lines
    for i in range(day+1,day+solving_period+1):
        if i < StartingDay + Planning_horizon:
            active_connections[i] = [b for b in active_lines if Lines[b]['EndDate']+1 == i]

    frozen_connections = {} #a dict that will contain connections between active lines and
        frozen lines. This will be used by the transit AC
    for d in range(day+1,day+solving_period+1):
        if d < StartingDay + Planning_horizon:
            frozen_connections[d] = [j for j in frozen_lines if Lines[j]['EndDate']+1 == d]

    #I use this connections dict to determine the longest possible routes, so that we can
        correctly determine which tasks need to be considered for planning
    Connections = {}
    for i in active_lines:
        Connections[i] = [j for j in active_lines if Lines[j]['StartDate'] == Lines[i]['
            EndDate']+1]
    #------------------------------------------------------------------------------------------
    #Determine the longest line combination so that we know which set of tasks to consider
    #the longest route that can be built differs between transit AC and starting AC
    routes = {}
    longest_flight_route= {}
    RouteCounter = 0
    for d in active_lines_starting_each_day:
        routes[d] = {}
        for i in active_lines_starting_each_day[d]:
            if len(Connections[i]) > 0:
                for i2 in Connections[i]:
                    if len(Connections[i2]) > 0:
                        for i3 in Connections[i2]:
                            if len(Connections[i3]) > 0:
                                for i4 in Connections[i3]:
                                    route = [i,i2,i3,i4]
                                    routelength = sum([Lines[b]['Flight Hours'] for b in
                                        route])
                                    routes[d][RouteCounter] = {'route':route, 'length':
                                        routelength}
                                    RouteCounter += 1
                            else:
                                route = [i,i2,i3]
                                routelength = sum([Lines[b]['Flight Hours'] for b in route])
                                routes[d][RouteCounter] = {'route':route, 'length':
                                    routelength}
                                RouteCounter += 1
                    else:
                        route = [i,i2]
                        routelength = sum([Lines[b]['Flight Hours'] for b in route])
                        routes[d][RouteCounter] = {'route':route, 'length':routelength}
                        RouteCounter += 1
            else:
                route = [i]
                routelength = sum([Lines[b]['Flight Hours'] for b in route])
                routes[d][RouteCounter] = {'route':route, 'length':routelength}
                RouteCounter += 1
        if len(routes[d]) > 0:
            longest_flight_route[d] = max([routes[d][b]['length'] for b in routes[d]]) #this
                is the longest route in FH that can be made with our active lines starting
                from each day
        else:
            longest_flight_route[d] = 0 #in these case there are no routes starting from a
                particular day.
    #
        ------------------------------------------------------------------------------------------
    if len(active_lines) > 0:
        line_latest_enddate = max([Lines[j]['EndDate'] for j in active_lines]) #also we need
            to know the latest enddate to determine all the possible tasks

    Encountered_lines = frozen_lines + active_lines #tasks can be planned after all
        encountered lines
     #make a dict containing all the encountered lines ending each day
    for d in range(StartingDay,StartingDay+Planning_horizon):
        Encountered_lines_ending_each_day[d] = [j for j in Lines_ending_each_day[d] if j in
            Encountered_lines]
    #this will be used in the capacity constraint for the non-active aircraft
    Frozen_line_endings_for_non_active_AC = {}
    for i in non_active_aircraft:
        Frozen_line_endings_for_non_active_AC[i] = {}
        for d in range(StartingDay,StartingDay+Planning_horizon):
            Frozen_line_endings_for_non_active_AC[i][d] = [j for j in Constructed_Routes[i]
                if Lines[j]['EndDate'] == d]

    #to create the constraint that each AC can only fly 1 active line at a time, we create a
        dict with the active lines that are being flown each day
    active_lines_flown_each_day = {}
    for d in range(day,day+solving_period):
        active_lines_flown_each_day[d] = [j for j in active_lines if Lines[j]['StartDate'] <=
             d and Lines[j]['EndDate'] >= d]

    #select the tasks that should be considered
    #to do this we need to acknowledge the flying history and the max possible length of the
        new lines
    Tasks = {}
    for TS in active_aircraft:
        Tasks[TS] = {}
        #We need to know when the aircraft will depart so that we can determine its maximum
            possible route
        if len(Constructed_Routes[TS]) < 1: #if no line has been assigned yet, we are at the
            starting day
            Departure_date_TS = StartingDay
        else:
            Departure_date_TS = Lines[Constructed_Routes[TS][-1]]['EndDate']+1 #select the
                arrival date of the most recently assigned line
        for index, row in TaskData[TS].iterrows():
            DueDate = row['Task Due']
            DueFH = row['FH Due']
            if DueDate <= (line_latest_enddate+1) or (DueFH - Flight_Hours_Recorded[TS])  <=
                longest_flight_route[Departure_date_TS]:
                task_number = int(row['Task Number'])
                Task = {'Task Number':task_number,
                    'TaskMH': row['Task MH'],
                    'Task Due Date' : row['Task Due'],
                    'Task FH Due' : row['FH Due']
                }
                Tasks[TS][task_number] = Task

    m = Model('RHM_Exact_Forecasting')

    My_X_Variables = tuplelist([])
    for i in range(len(active_aircraft)):
        for j in active_lines:  #only active lines need to be distributed, frozen lines have
            already been distributed in a previous iteration
            My_X_Variables.append((i,j))
    My_Y_Variables = tuplelist([])
    for i in range(len(active_aircraft)):
        for t in Tasks[active_aircraft[i]]:
            for j in Encountered_lines:  #an AC can plan maintenance both after an active
                line and after a frozen line
                My_Y_Variables.append((i,t,j))
    #the following help variables are used to calculate the number of lost flying hours. This
         was necessary because a MIN or MAX function only works in a constraint and not in an
         objective function in Gurobi
    LFH1_ac = {}
    LFH2_ac = {}
    LFH3_ac = {}
    LFH1_fr = {} #we define help variables for frozen lines and active lines independently
    LFH2_fr = {}
    LFH3_fr = {}
    for i in range(len(active_aircraft)):
        for t in Tasks[active_aircraft[i]]:
            for j in Encountered_lines:
                LFH1_ac[i,t,j] = m.addVar(lb = -10000, ub = 10000)
                LFH2_ac[i,t,j] = m.addVar(lb = -10000, ub = 10000)
                LFH3_ac[i,t,j] = m.addVar(lb = -10000, ub = 10000)
                LFH1_fr[i,t,j] = m.addVar(lb = -10000, ub = 10000)
                LFH2_fr[i,t,j] = m.addVar(lb = -10000, ub = 10000)
                LFH3_fr[i,t,j] = m.addVar(lb = -10000, ub = 10000)
    #this is for the non_active aircraft
    for i in range(len(active_aircraft),len(all_aircraft)):
        for t in non_active_task_set[all_aircraft[i]]: #the set of tasks that non-active AC
            have to schedule has been determined in a previous iteration
            for j in Constructed_Routes[all_aircraft[i]]: #non-active AC i can only plan
                after lines that it has flown
                My_Y_Variables.append((i,t,j))
                LFH1_fr[i,t,j] = m.addVar(lb = -10000, ub = 10000)
                LFH2_fr[i,t,j] = m.addVar(lb = -10000, ub = 10000)
                LFH3_fr[i,t,j] = m.addVar(lb = -10000, ub = 10000)

    x = m.addVars(My_X_Variables, vtype=GRB.BINARY, name='x')   # if line j is assigned to AC
        i
    y = m.addVars(My_Y_Variables, vtype=GRB.BINARY, name='y')   # equals 1 if AC i plans task
        t AFTER line j

    m.setObjective(quicksum(y[i,t,j]*Tasks[active_aircraft[i]][t]['TaskMH']*LFH3_ac[i,t,j]
        for i in range(len(active_aircraft)) for j in active_lines for t in Tasks[
        active_aircraft[i]]) + quicksum(y[i,t,j]*Tasks[active_aircraft[i]][t]['TaskMH']*
        LFH3_fr[i,t,j] for i in range(len(active_aircraft)) for j in frozen_lines for t in
        Tasks[active_aircraft[i]]) + quicksum(y[i,t,j]*AllTaskData[all_aircraft[i]][t]['
        TaskMH']*LFH3_fr[i,t,j] for i in range(len(active_aircraft),len(all_aircraft)) for j
        in Constructed_Routes[all_aircraft[i]] for t in non_active_task_set[all_aircraft[i]])
         , GRB.MINIMIZE)

#----- Routing Section
    #every line must be flown exactly once
    m.addConstrs(quicksum(x[i,j] for i in range(len(active_aircraft))) == 1 for j in
        active_lines)
    #each first day active AC needs to fly exactly 1 first day active line
    m.addConstrs(quicksum(x[i,j] for j in starting_active_lines) == 1 for i in range(len(
        starting_aircraft)))
    #for the first day we can only choose a feasible starting line
    if day == StartingDay:
        m.addConstrs(quicksum(x[i,j] for j in PossibleStartLines[active_aircraft[i]]) == 1
            for i in range(len(active_aircraft)))
    #now the connection constraint
    for i in range(len(active_aircraft)):
        for d in range(day+1,day+solving_period):
            if d < StartingDay + Planning_horizon:
                m.addConstr(quicksum(x[i,j] for j in active_lines_starting_each_day[d]) ==
                    quicksum(x[i,j] for j in active_connections[d]) + quicksum(HAFL[
                    active_aircraft[i]][p] for p in frozen_connections[d]))
    #An AC can fly at most 1 line at a time: this constraint is not required to converge the
        solution but it was found to speed up the calculations quite a bit
    for d in range(day,day+solving_period):
        m.addConstrs(quicksum(x[i,j] for j in active_lines_flown_each_day[d] ) <= 1 for i in
            range(len(active_aircraft)))

#----- Task Planning
    #Every considered task can be planned at most once, after an active line or a frozen line
    m.addConstrs(quicksum(y[i,t,j] for j in Encountered_lines) <= 1 for i in range(len(
        active_aircraft)) for t in Tasks[active_aircraft[i]])
    #for non-active AC: all their tasks must be planned
    m.addConstrs(quicksum(y[i,t,j] for j in Constructed_Routes[all_aircraft[i]]) == 1 for i
        in range(len(active_aircraft),len(all_aircraft)) for t in non_active_task_set[
        all_aircraft[i]])

    #All tasks in T_i which are due before a due date should be planned if that due date is
```

```
        closer than the end of the selected line
    m.addConstrs(−1000*quicksum(y[i,t,j]*MS[j] for j in Encountered_lines) <= Tasks[
        active_aircraft[i]][t]['Task Due Date'] − (2+quicksum([x[i,j]*Lines[j]['EndDate'] for
         j in ending_lines])) for i in range(len(active_aircraft)) for t in Tasks[
        active_aircraft[i]])

    #All tasks in T_i which are due before X flight hours should be planned if the selected
        route is longer than X flight hours.
    m.addConstrs(−1000*quicksum(y[i,t,j]*MS[j] for j in Encountered_lines) <= (Tasks[
        active_aircraft[i]][t]['Task FH Due'] − (Flight_Hours_Recorded[active_aircraft[i]] +
        quicksum(x[i,j]*Lines[j]['Flight Hours'] for j in active_lines))) for i in range(len(
        active_aircraft)) for t in Tasks[active_aircraft[i]] )

    #If task t of AC i is scheduled, it must be scheduled before the due date:
    m.addConstrs(y[i,t,j]*Lines[j]['EndDate']+1 <= Tasks[active_aircraft[i]][t]['Task Due
        Date'] for i in range(len(active_aircraft)) for t in Tasks[active_aircraft[i]] for j
        in Encountered_lines)
    #For non active AC as well:
    m.addConstrs(y[i,t,j]*Lines[j]['EndDate']+1 <= AllTaskData[all_aircraft[i]][t]['Task Due
        Date'] for i in range(len(active_aircraft),len(all_aircraft)) for t in
        non_active_task_set[all_aircraft[i]] for j in Constructed_Routes[all_aircraft[i]])

    #If task t of AC i is scheduled, it must be scheduled before the due flight hours:
    #This constraint is broken up in the frozen lines and the active lines, as they require
        different formulations
    for i in range(len(active_aircraft)):
        for t in Tasks[active_aircraft[i]]:
            #frozen lines:
            m.addConstrs(y[i,t,j]*Flown_FH_after_line[j]  <= Tasks[active_aircraft[i]][t]['
                Task FH Due'] for j in frozen_lines)
            #active lines:
            m.addConstrs(y[i,t,j]*(Flight_Hours_Recorded[active_aircraft[i]] + quicksum(x[i,j
                ]*Lines[j]['Flight Hours'] for d in range(day,Lines[j]['StartDate']+1) for j
                in active_lines_starting_each_day[d])) <= Tasks[active_aircraft[i]][t]['Task
                FH Due'] for j in active_lines)
    #for non−active AC as well:
    m.addConstrs(y[i,t,j]*Flown_FH_after_line[j]  <= AllTaskData[all_aircraft[i]][t]['Task FH
         Due'] for i in range(len(active_aircraft),len(all_aircraft)) for t in
        non_active_task_set[all_aircraft[i]] for j in Constructed_Routes[all_aircraft[i]])

    #Only plan a task after a line if the AC has flown that line
    #Again we need to break up between frozen lines and active lines
    for i in range(len(active_aircraft)):
       #frozen lines
        m.addConstrs(len(Tasks[active_aircraft[i]])*HAFL[active_aircraft[i]][j] >= quicksum(y
            [i,t,j] for t in Tasks[active_aircraft[i]]) for j in frozen_lines)
        #active lines
        m.addConstrs(len(Tasks[active_aircraft[i]])*x[i,j] >= quicksum(y[i,t,j] for t in
            Tasks[active_aircraft[i]]) for j in active_lines)

    #Capacity constraint:
    for d in range(StartingDay,StartingDay+Planning_horizon):
        m.addConstr(quicksum(y[i,t,j]*Tasks[active_aircraft[i]][t]['TaskMH'] for i in range(
            len(active_aircraft)) for j in Encountered_lines_ending_each_day[d] for t in
            Tasks[active_aircraft[i]]) + quicksum(y[i,t,j]*AllTaskData[all_aircraft[i]][t]['
            TaskMH'] for i in range(len(active_aircraft),len(all_aircraft)) for j in
            Frozen_line_endings_for_non_active_AC[all_aircraft[i]][d] for t in
            non_active_task_set[all_aircraft[i]]) <= Station_Capacity)

    #Calculate the lost flying hours to help the objective function
    #active lines
    for i in range(len(active_aircraft)):
        for j in active_lines: #for the active lines
            for t in Tasks[active_aircraft[i]]:
                m.addConstr(LFH1_ac[i,t,j] == Tasks[active_aircraft[i]][t]['Task FH Due']− (
                    Flight_Hours_Recorded[active_aircraft[i]] + Lines[j]['Flight Hours'] +
                    quicksum(x[i,k]*Lines[k]['Flight Hours'] for d in range(day,Lines[j]['
                    StartDate']) for k in active_lines_starting_each_day[d]))) #lost flying
                    hours due to task being executed before remaining legal FH
                m.addConstr(LFH2_ac[i,t,j] == average_FH_per_day*(Tasks[active_aircraft[i]][t
                    ]['Task Due Date'] −(Lines[j]['EndDate']+1))) #Lost flying hours due to
                task being executed before its due date
            m.addConstr(LFH3_ac[i,t,j] == min_(LFH1_ac[i,t,j],LFH2_ac[i,t,j]))
    for j in frozen_lines: #for the frozen lines
        for t in Tasks[active_aircraft[i]]:
            m.addConstr(LFH1_fr[i,t,j] == Tasks[active_aircraft[i]][t]['Task FH Due'] -
                Flown_FH_after_line[j]) #lost flying hours due to task being executed
                before remaining legal FH
            m.addConstr(LFH2_fr[i,t,j] == average_FH_per_day*(Tasks[active_aircraft[i]][t
                ]['Task Due Date'] -(Lines[j]['EndDate']+1))) #Lost flying hours due to
                task being executed before its due date
            m.addConstr(LFH3_fr[i,t,j] == min_(LFH1_fr[i,t,j],LFH2_fr[i,t,j]))
#for non-active aircraft as well
for i in range(len(active_aircraft),len(all_aircraft)):
    for j in Constructed_Routes[all_aircraft[i]]:
        for t in non_active_task_set[all_aircraft[i]]:
            m.addConstr(LFH1_fr[i,t,j] == AllTaskData[all_aircraft[i]][t]['Task FH Due']
                - Flown_FH_after_line[j]) #lost flying hours due to task being executed
                before remaining legal FH
            m.addConstr(LFH2_fr[i,t,j] == average_FH_per_day*(AllTaskData[all_aircraft[i
                ]][t]['Task Due Date'] -(Lines[j]['EndDate']+1))) #Lost flying hours due
                to task being executed before its due date
            m.addConstr(LFH3_fr[i,t,j] == min_(LFH1_fr[i,t,j],LFH2_fr[i,t,j]))

m.optimize()
#Save routing solution and prepare for next day iteration
line_solution = m.getAttr('x', x)
Routing_Solution = [(i,j) for (i,j) in line_solution if line_solution[i,j] > 0.9]
task_solution = m.getAttr('x', y)
Maintenance_Planning_Solution = [(i,t,j) for (i,t,j) in task_solution if task_solution[i,
    t,j] > 0.9]

#i need to update: frozen_lines, Flight_hours_recorded, constructed_route, HAFL and
    active_aircraft list
#I only want to freeze the solutions for the lines that departed today, so not for future
     lines
for i,j in Routing_Solution:
    if Lines[j]['StartDate'] == day:
        frozen_lines.append(j) #update the frozen lines
        HAFL[active_aircraft[i]][j] = 1 #update HAFL
        Constructed_Routes[active_aircraft[i]].append(j) #update Constructed Routes
        Flight_Hours_Recorded[active_aircraft[i]] += Lines[j]['Flight Hours'] #update the
             recorded flight hours

for i in all_aircraft:
    Most_Recent_Maint_Schedule[i] = []    #reset the maint schedule of the active AC so
        that we can fill it in again

for i,t,j in Maintenance_Planning_Solution:
    if Lines[j]['StartDate'] <= day:
        Most_Recent_Maint_Schedule[all_aircraft[i]].append([t,j])

#Now I need to specify how many flight hours were flown after each line in the frozen
    lines, for future planning needs
for i in TailNumbers: #for every AC
    for m in range(len(Constructed_Routes[i])): #for every line that it has been assigned
         so far
        the_line = Constructed_Routes[i][m] #identify the line
        flown_lines_before_it = Constructed_Routes[i][:(m+1)] #identify which lines are
            assigned to the AC before the considered line
        Flown_FH_after_line[the_line] = sum([Lines[b]['Flight Hours'] for b in
            flown_lines_before_it]) #determine how many FH are flown before the line
            starts

#Keep track of the current scores of the routes of each AC
for i in all_aircraft:
    MaintScore[i]=0 #reset the score for the active AC cause we are going to recalculate
        them
    for t,j in Most_Recent_Maint_Schedule[i]: #iterate through the planned maintenance
         for AC i
        Score_days = (AllTaskData[i][t]['Task Due Date'] - (Lines[j]['EndDate']+1))*
            average_FH_per_day #Lost flying hours due to task being executed before its
```

```
                due date
            Score_FH = AllTaskData[i][t]['Task FH Due'] - Flown_FH_after_line[j] #lost flying
                 hours due to task being executed before remaining legal FH
            overall_score = AllTaskData[i][t]['TaskMH']*min(Score_days,Score_FH)
            MaintScore[i] += overall_score

    #Now to determine the new starting AC and transit AC for the next iteration
    active_aircraft = []
    starting_aircraft = []
    transit_aircraft = []
    non_active_aircraft = []
    for i in TailNumbers:
        Line_end = Lines[Constructed_Routes[i][-1]]['EndDate']
        if Line_end < StartingDay + Planning_horizon and Line_end < StartingDay +
            Planning_horizon - 1:
            if Line_end == day:
                starting_aircraft.append(i)
            if Line_end > day and Line_end < day+solving_period:
                transit_aircraft.append(i)
    active_aircraft = starting_aircraft + transit_aircraft
    for i in TailNumbers:
        if i not in active_aircraft:
            non_active_aircraft.append(i)
    all_aircraft = active_aircraft + non_active_aircraft
    #determine the tasks that the non-active AC have planned and will need to replan next
        iteration
    for i in non_active_aircraft:
        non_active_task_set[i] = [t for (t,j) in Most_Recent_Maint_Schedule[i]]

#Calculate final score
print('Final Scores: '+str(MaintScore))
print('Final Total Score: '+str(sum(MaintScore.values())))
stopwatch_end = time.time()-stopwatch_start
print('Total Elapsed time: '+str(stopwatch_end))
print('Total number of AC: ' +str(len(TailNumbers)))
```

G
Python Code - Rolling Horizon Matheuristic with Average Flight Hour Forecasting

```
# -*- coding: utf-8 -*-
"""
Created on Tue May 12 10:19:54 2020

@author: Alex
in this script we solve the RHM with average flight-hour forecasting. The number of days we
    forecast for is given as an input parameter.
"""

import pandas as pd
from Line_extraction import Lines_Only #this is a function that will extract lines from
    flightdata
from gurobipy import *
import copy
import time

TaskData = pd.read_excel('TaskData_20_tasks.xlsx', sheet_name=None)
FlightData = pd.read_excel('FlightData.xlsx', sheet_name=None)
TailNumbers = pd.ExcelFile('FlightData.xlsx').sheet_names
TailNumbers = ['PH-BGF','PH-BGG','PH-BGH','PH-BGI','PH-BGK'] #this list contains all the tail
     numbers that are considered for the problem

#Parameters
Planning_horizon = 7
StartingDay = 43831 #Excel uses a number for a date, which works very well. 43831 translates
    to 1-1-2020
Station_Capacity = 100 #in man-hours per night
forecasting_period = 1   #number of days that we forecast for

#extract the lines from the flightdata. Within the function Lines_Only also a due date for
    the first standard check is randomly generated
Lines, StartLines, EndLines,standard_check_due_date = Lines_Only(FlightData,TailNumbers)
#calculate the average flight hours flown per day
average_FH_per_day = round(sum([Lines[b]['Flight Hours'] for b in Lines])/(len(TailNumbers)*
    Planning_horizon),1)

#I make a dictionary here containing all the task data to which i can always refer in the
    code
AllTaskData = {}
for i in TailNumbers:
    AllTaskData[i] = {}
    for index, row in TaskData[i].iterrows():
        task_number = int(row['Task Number'])
        Task = {'Task Number':task_number,
            'TaskMH': row['Task MH'],
            'Task Due Date' : row['Task Due'],
            'Task FH Due' : row['FH Due']
        }
        AllTaskData[i][task_number] = Task

#%% Preprocessing for building the routes
#Here multiple dictionaries are built containing line information, which are used when
    defining the model
Lines_starting_each_day = {}
Lines_ending_each_day = {}
for i in range(Planning_horizon):
    for j in Lines:
       Lines_starting_each_day[StartingDay+i] = [a for a in Lines if Lines[a]['StartDate'] ==
            StartingDay+i]
       Lines_ending_each_day[StartingDay+i] = [a for a in Lines if Lines[a]['EndDate'] ==
           StartingDay+i]

#In the given flight data Amsterdam is the maintenance station
#Whether or not Lines end on amsterdam. this is necessary for the ending lines, which could
    end on different stations than Amsterdam.
MS = {} #MS: Maintenance station
for j in Lines:
    if Lines[j]['Arrival'] == 'Amsterdam':
        MS[j] = 1
    else:
        MS[j] = 0

#Determine the starting points for each tail
StartPoint = {}
for i in range(len(TailNumbers)):
    Starter = StartLines[i]
    #the first Startline is from the first aircraft and so on
    StartPoint[TailNumbers[i]] = Lines[Starter]['Departure']
#Now from the startpoint determine the set of 1st day Lines possible for each tailsign
PossibleStartLines = {}
for i in TailNumbers:
    PossibleStartLines[i] = [j for j in StartLines if Lines[j]['Departure'] == StartPoint[i]
        and Lines[j]['EndDate'] < standard_check_due_date[i]]

#%% Algorithm Start
#The encountered lines are divided in frozen lines and active lines. frozen lines have been
    assigned to aircraft already. active lines will be assigned during this iteration
#there are 3 types of aircraft: 1. starting aircraft start from the first day of the central
    period and their lines will be frozen
#2. transit AC will start a new line during the forecasting period but not during the central
     period
#3. non-active AC will not start a new line during either the central or forecasting period
starting_aircraft = copy.deepcopy(TailNumbers) #starting AC are the AC that will be assigned
    a line that will be frozen at the end of the iteration
transit_aircraft = [] #transit AC are AC that will be assigned a line in the forecasting
    period, but not one in the central period
active_aircraft = starting_aircraft + transit_aircraft
non_starting_aircraft = [] #AC that will not start a new line in the forecasting period
all_aircraft = starting_aircraft + non_starting_aircraft
non_starting_task_set = {} #the non_starting_AC will have to replan their already planned
    maintenance every iteration to make sure we comply with the capacity constraint
Flight_Hours_Recorded = {} #a dict that will save the number of FH each AC has flown
    throughout the iterations
frozen_lines = [] #a list of all lines that have been assigned and frozen
Constructed_Routes = {} #in this dict we will save the frozen routes
Most_Recent_Maint_Schedule = {} #a dict that holds the maintenance schedule, which is updated
     during every iteration
Flown_FH_after_line = {} #this dict will save the number of FH that are recorded after
    completing a frozen line
Encountered_lines_ending_each_day = {}
HAFL = {} #a dictionary that holds information on which frozen line has been assigned to
    which AC. HAFL: Has AC flown line?
MaintScore = {} #a dict that will save the scores of the maintenance planning for each AC
for i in TailNumbers:
    Flight_Hours_Recorded[i] = 0
    Constructed_Routes[i] = []
    Most_Recent_Maint_Schedule[i] = []
    MaintScore[i] = 0
    HAFL[i] = {}
    for j in Lines:
        HAFL[i][j] = 0

stopwatch_start = time.time() #start a timer

for day in range(StartingDay,StartingDay+Planning_horizon):

    #Select the lines starting at this day
    starting_lines = [b for b in Lines if Lines[b]['StartDate'] == day]
    if len(starting_lines) > 0:
        longest_line_FH = max([Lines[b]['Flight Hours'] for b in starting_lines]) #the
            longest line in FH will help determine which tasks should be considered
        line_latest_enddate = max([Lines[b]['EndDate'] for b in starting_lines]) #the latest
            line enddate will help determine which tasks should be considered

    Encountered_lines = frozen_lines + starting_lines #we can plan maintenance after both
        frozen and active lines
    #make a dict containing all the encountered lines ending each day
    for d in range(StartingDay,StartingDay+Planning_horizon):
        Encountered_lines_ending_each_day[d] = [j for j in Lines_ending_each_day[d] if j in
            Encountered_lines]
    #this will be used in the capacity constraint for the non−starting aircraft
    Frozen_line_endings_for_non_starting_AC = {}
    for i in non_starting_aircraft:
        Frozen_line_endings_for_non_starting_AC[i] = {}
        for d in range(StartingDay,StartingDay+Planning_horizon):
            Frozen_line_endings_for_non_starting_AC[i][d] = [j for j in Constructed_Routes[i]
                 if Lines[j]['EndDate'] == d]

    #select the tasks that should be considered
    #to do this we need to acknowledge the flying history and the max possible length of the
        new lines
    Tasks = {}
    for TS in starting_aircraft:
        Tasks[TS] = {}
        for index, row in TaskData[TS].iterrows():
            DueDate = row['Task Due']
            DueFH = row['FH Due']
            if DueDate <= (line_latest_enddate+1) or (DueFH − Flight_Hours_Recorded[TS])  <=
                longest_line_FH:
                task_number = int(row['Task Number'])
                Task = {'Task Number':task_number,
                    'TaskMH': row['Task MH'],
                    'Task Due Date' : row['Task Due'],
                    'Task FH Due' : row['FH Due']
                }
                Tasks[TS][task_number] = Task

    #−−−−−−−−−−−−−−−−−− Forecast preparation −−−−−−−−−−−−−−−
    #Determine the future lines that are to be considered within the forecasting period
    Considered_Future_Lines = {}
    Considered_future_lines_listed = []
    for i in range(day+1,day+forecasting_period+1):
        Considered_Future_Lines[i] = [b for b in Lines if Lines[b]['StartDate'] == i] #all
            the lines starting tomorrow
        Considered_future_lines_listed.extend([b for b in Lines if Lines[b]['StartDate'] == i
            ])

    #there are 3 types of connections to the future flights, active lines, already assigned
        lines, and future lines
    #all lines starting on the same day have the same connections, so we group them by day
        and not by individual line
    active_connections = {}
    frozen_connections = {}
    future_connections = {}
    for d in range(day+1,day+forecasting_period+1):
        if d < StartingDay + Planning_horizon:
            active_connections[d] = [j for j in starting_lines if Lines[j]['EndDate']+1 == d]
            frozen_connections[d] = [j for j in frozen_lines if Lines[j]['EndDate']+1 == d]
            future_connections[d] = [j for j in Considered_future_lines_listed if Lines[j]['
                EndDate']+1 == d]

    #a cost prediction will be made for every AC i if that AC is assigned to fly future line
        j
    C = {}
    for i in range(len(active_aircraft)): #for each AC that can be assigned a future line
        for j in Considered_future_lines_listed: #for every future line
            #the tasks that AC i will have to move forward if future line j is chosen
            #predict OR determine the number of FH flown at the start of the line
            if len(Constructed_Routes[active_aircraft[i]]) >= 1:
                PredictedFH_at_start_of_future_line = Flight_Hours_Recorded[active_aircraft[i
                    ]] + (Lines[j]['StartDate'] − Lines[Constructed_Routes[active_aircraft[i
                    ]][−1]]['EndDate']−1)*average_FH_per_day #the predicted FH that the AC
                    has travelled right before starting the future line in question
            else: #during the first iteration we dont have a constructed route yet
                PredictedFH_at_start_of_future_line = Flight_Hours_Recorded[active_aircraft[i
                    ]] + (Lines[j]['StartDate'] − day)*average_FH_per_day
            PredictedFH_at_end_of_future_line = PredictedFH_at_start_of_future_line + (Lines[
                j]['EndDate']+1−Lines[j]['StartDate'])*average_FH_per_day #the predicted FH
                that the AC has travelled after completion of the future line in question

            Penalty = 0
            for index, row in TaskData[active_aircraft[i]].iterrows():
                DueDate = row['Task Due']
                DueFH = row['FH Due']
                if DueDate > Lines[j]['StartDate'] and DueFH >
                    PredictedFH_at_start_of_future_line: #so the task has not been executed
                    yet
                    if DueDate <= Lines[j]['EndDate'] or DueFH <
                        PredictedFH_at_end_of_future_line: #if the task is expected to go due
                         within the line
                        #each task will have a cost because its moved up to j. It is the goal
                             to sum up these costs
                        Penalty_FH = row['FH Due']-PredictedFH_at_start_of_future_line
                        Penalty_Day = (row['Task Due']-Lines[j]['StartDate'])*
                            average_FH_per_day
                        Penalty += row['Task MH']*min(Penalty_FH,Penalty_Day)
            C[i,j] = Penalty

    m = Model('RHM_Average_FH_Forecasting')

    My_X_Variables = tuplelist([])
    for i in range(len(active_aircraft)):
        for j in starting_lines: #frozen lines have already been distributed and for future
            lines we have a seperate variable
            My_X_Variables.append((i,j))
    My_Y_Variables = tuplelist([])
    for i in range(len(starting_aircraft)):
        for t in Tasks[starting_aircraft[i]]:
            for j in Encountered_lines:  #We should be able to plan anywhere
                My_Y_Variables.append((i,t,j))
    My_Q_Variables = tuplelist([])
    for i in range(len(active_aircraft)):
        for k in Considered_future_lines_listed:
            My_Q_Variables.append((i,k))  #will equal 1 if AC i will fly future line k
            #i use k instead of j to make a clearer distinction between starting lines and
                future lines
    LFH1_ac = {}
    LFH2_ac = {} #_ac = active lines
    LFH3_ac = {}
    LFH1_fr = {} #we define help variables for frozen lines and active lines independently
    LFH2_fr = {} #_ fr = frozen lines
    LFH3_fr = {}
    for i in range(len(starting_aircraft)):
        for t in Tasks[starting_aircraft[i]]:
            for j in Encountered_lines:
                LFH1_ac[i,t,j] = m.addVar(lb = -10000, ub = 10000)
                LFH2_ac[i,t,j] = m.addVar(lb = -10000, ub = 10000)
                LFH3_ac[i,t,j] = m.addVar(lb = -10000, ub = 10000)
                LFH1_fr[i,t,j] = m.addVar(lb = -10000, ub = 10000)
                LFH2_fr[i,t,j] = m.addVar(lb = -10000, ub = 10000)
                LFH3_fr[i,t,j] = m.addVar(lb = -10000, ub = 10000)
    #this is for the non_starting aircraft
    for i in range(len(starting_aircraft),len(all_aircraft)):
        for t in non_starting_task_set[all_aircraft[i]]: #the set of tasks that non-starting
            AC have to schedule has been determined in a previous iteration
            for j in Constructed_Routes[all_aircraft[i]]: #non-starting AC i can only plan
                after lines that it has flown
                My_Y_Variables.append((i,t,j))
                LFH1_fr[i,t,j] = m.addVar(lb = -10000, ub = 10000)
                LFH2_fr[i,t,j] = m.addVar(lb = -10000, ub = 10000)
                LFH3_fr[i,t,j] = m.addVar(lb = -10000, ub = 10000)

    x = m.addVars(My_X_Variables, vtype=GRB.BINARY, name='x')  # equals 1 if line j is
        assigned to AC i, 0 otherwise
    y = m.addVars(My_Y_Variables, vtype=GRB.BINARY, name='y')  # equals 1 if AC i plans task
        t AFTER line j, 0 otherwise
    q = m.addVars(My_Q_Variables, vtype=GRB.BINARY, name='q')  # equals 1 AC i is assigned to
         fly future line k, 0 otherwise

    #LFH = Lost Flying Hours
    #the objective function is built of 4 parts: minimize LFH for starting AC who plan after
        starting lines, minimze LFH for starting AC after frozen lines, minimze LFH for non-
        starting AC and minimize predicted future cost
    #So it is actually just 2 parts: minimze current planning LFH and minimze predicted
        future costs. But the current planning LFH required different formulations
    m.setObjective(quicksum(y[i,t,j]*Tasks[starting_aircraft[i]][t]['TaskMH']*LFH3_ac[i,t,j]
        for i in range(len(starting_aircraft)) for j in starting_lines for t in Tasks[
        starting_aircraft[i]]) + quicksum(y[i,t,j]*Tasks[starting_aircraft[i]][t]['TaskMH']*
        LFH3_fr[i,t,j] for i in range(len(starting_aircraft)) for j in frozen_lines for t in
        Tasks[starting_aircraft[i]]) + quicksum(y[i,t,j]*AllTaskData[all_aircraft[i]][t]['
        TaskMH']*LFH3_fr[i,t,j] for i in range(len(starting_aircraft),len(all_aircraft)) for
        j in Constructed_Routes[all_aircraft[i]] for t in non_starting_task_set[all_aircraft[
        i]]) + quicksum(q[i,k]*C[i,k] for i in range(len(active_aircraft)) for k in
        Considered_future_lines_listed) , GRB.MINIMIZE)

#----- Routing Section
    #every active line must be flown
    m.addConstrs(quicksum(x[i,j] for i in range(len(starting_aircraft))) == 1 for j in
        starting_lines)
    #Every active AC must fly exactly one active line
    m.addConstrs(quicksum(x[i,j] for j in starting_lines) == 1 for i in range(len(
        starting_aircraft)))
    #for the first day we can only choose a feasible starting line
    if day == StartingDay:
       m.addConstrs(quicksum(x[i,j] for j in PossibleStartLines[starting_aircraft[i]]) == 1
           for i in range(len(starting_aircraft)))

#----- Task Planning
    #Every considered task can be planned at most once, after an active line or a frozen line
    m.addConstrs(quicksum(y[i,t,j] for j in Encountered_lines) <= 1 for i in range(len(
        starting_aircraft)) for t in Tasks[starting_aircraft[i]])
    #for non-starting AC: all their tasks must be planned
    m.addConstrs(quicksum(y[i,t,j] for j in Constructed_Routes[all_aircraft[i]]) == 1 for i
        in range(len(starting_aircraft),len(all_aircraft)) for t in non_starting_task_set[
        all_aircraft[i]])

    #All tasks in T_i which are due before a due date should be planned if that due date is
        closer than the end of the selected line
    m.addConstrs(-1000*quicksum(y[i,t,j]*MS[j] for j in Encountered_lines) <= Tasks[
        starting_aircraft[i]][t]['Task Due Date'] - (quicksum(x[i,j]*Lines[j]['EndDate'] for
        j in starting_lines)+2) for i in range(len(starting_aircraft)) for t in Tasks[
        starting_aircraft[i]])

    #All tasks in T_i which are due before X flight hours should be planned if the selected
        route is longer than X flight hours.
    m.addConstrs(-1000*quicksum(y[i,t,j]*MS[j] for j in Encountered_lines) <= (Tasks[
        starting_aircraft[i]][t]['Task FH Due'] - (Flight_Hours_Recorded[starting_aircraft[i
        ]] + quicksum(x[i,j]*Lines[j]['Flight Hours'] for j in starting_lines))) for i in
        range(len(starting_aircraft)) for t in Tasks[starting_aircraft[i]])

    #If task t of AC i is scheduled, it must be scheduled before the due date:
    m.addConstrs(y[i,t,j]*Lines[j]['EndDate']+1 <= Tasks[starting_aircraft[i]][t]['Task Due
        Date'] for i in range(len(starting_aircraft)) for t in Tasks[starting_aircraft[i]]
        for j in Encountered_lines)
    #For non starting AC as well:
    m.addConstrs(y[i,t,j]*Lines[j]['EndDate']+1 <= AllTaskData[all_aircraft[i]][t]['Task Due
        Date'] for i in range(len(starting_aircraft),len(all_aircraft)) for t in
        non_starting_task_set[all_aircraft[i]] for j in Constructed_Routes[all_aircraft[i]])

    #If task t of AC i is scheduled, it must be scheduled before the due flight hours:
    #This constraint is broken up in the frozen lines and the active lines, as they require
        different formulations
    for i in range(len(starting_aircraft)):
        for t in Tasks[starting_aircraft[i]]:
            #frozen lines:
            m.addConstrs(y[i,t,j]*Flown_FH_after_line[j]  <= Tasks[starting_aircraft[i]][t]['
                Task FH Due'] for j in frozen_lines)
            #active lines:
            m.addConstrs(y[i,t,j]*(Flight_Hours_Recorded[starting_aircraft[i]] + Lines[j]['
                Flight Hours']) <= Tasks[starting_aircraft[i]][t]['Task FH Due'] for j in
                starting_lines)
    #for non-starting AC as well:
    m.addConstrs(y[i,t,j]*Flown_FH_after_line[j]  <= AllTaskData[all_aircraft[i]][t]['Task FH
         Due']  for  i  in  range(len(starting_aircraft),len(all_aircraft))  for  t  in
         non_starting_task_set[all_aircraft[i]]  for  j  in  Constructed_Routes[all_aircraft[i]])

    #Only plan a task after a line if the AC has flown that line
    #Again we need to break up between frozen lines and active lines
    for i in range(len(starting_aircraft)):
       #frozen lines
        m.addConstrs(len(Tasks[starting_aircraft[i]])*HAFL[starting_aircraft[i]][j] >=
            quicksum(y[i,t,j] for t in Tasks[starting_aircraft[i]]) for j in frozen_lines)
        #active lines
        m.addConstrs(len(Tasks[starting_aircraft[i]])*x[i,j] >= quicksum(y[i,t,j] for t in
            Tasks[starting_aircraft[i]]) for j in starting_lines)

    #Capacity constraint:
    for d in range(StartingDay,StartingDay+Planning_horizon):
        m.addConstr(quicksum(y[i,t,j]*Tasks[active_aircraft[i]][t]['TaskMH'] for i in range(
            len(starting_aircraft)) for j in Encountered_lines_ending_each_day[d] for t in
            Tasks[starting_aircraft[i]]) + quicksum(y[i,t,j]*AllTaskData[all_aircraft[i]][t][
            'TaskMH'] for i in range(len(starting_aircraft),len(all_aircraft)) for j in
            Frozen_line_endings_for_non_starting_AC[all_aircraft[i]][d] for t in
            non_starting_task_set[all_aircraft[i]]) <= Station_Capacity)

    #Calculate the lost flying hours to help the objective function
    #active lines
    for i in range(len(starting_aircraft)):
        for j in starting_lines:
            for t in Tasks[starting_aircraft[i]]:
                m.addConstr(LFH1_ac[i,t,j] == Tasks[starting_aircraft[i]][t]['Task FH Due']-
                    (Flight_Hours_Recorded[starting_aircraft[i]] + Lines[j]['Flight Hours']))
                m.addConstr(LFH2_ac[i,t,j] == average_FH_per_day*(Tasks[starting_aircraft[i
                    ]][t]['Task Due Date'] -(Lines[j]['EndDate']+1)))
                m.addConstr(LFH3_ac[i,t,j] == min_(LFH1_ac[i,t,j],LFH2_ac[i,t,j]))
        for j in frozen_lines:
            for t in Tasks[starting_aircraft[i]]:
                m.addConstr(LFH1_fr[i,t,j] == Tasks[starting_aircraft[i]][t]['Task FH Due'] -
                     Flown_FH_after_line[j])
                m.addConstr(LFH2_fr[i,t,j] == average_FH_per_day*(Tasks[starting_aircraft[i
                    ]][t]['Task Due Date'] -(Lines[j]['EndDate']+1)))
                m.addConstr(LFH3_fr[i,t,j] == min_(LFH1_fr[i,t,j],LFH2_fr[i,t,j]))
    #for non-starting aircraft as well
    for i in range(len(starting_aircraft),len(all_aircraft)):
        for j in Constructed_Routes[all_aircraft[i]]:
            for t in non_starting_task_set[all_aircraft[i]]:
                m.addConstr(LFH1_fr[i,t,j] == AllTaskData[all_aircraft[i]][t]['Task FH Due']
                    - Flown_FH_after_line[j]) #lost flying hours due to task being executed
                    before remaining legal FH
                m.addConstr(LFH2_fr[i,t,j] == average_FH_per_day*(AllTaskData[all_aircraft[i
                    ]][t]['Task Due Date'] -(Lines[j]['EndDate']+1))) #Lost flying hours due
                    to task being executed before its due date
                m.addConstr(LFH3_fr[i,t,j] == min_(LFH1_fr[i,t,j],LFH2_fr[i,t,j]))

    #FORECAST Constraints
    #Forecasting constraints should not be included for the last day because they do not make
         sense and even give errors
    if day < (StartingDay + Planning_horizon-1):
        #Every future line should be assigned exactly once:
        m.addConstrs(quicksum(q[i,k] for i in range(len(active_aircraft))) == 1 for k in
            Considered_future_lines_listed)
        #the connection constraint should force every considered future line to be assigned
            to an AC
        for i in range(len(active_aircraft)):
            for d in range(day+1,day+forecasting_period+1):
                if d < StartingDay + Planning_horizon:
                    m.addConstr(quicksum(q[i,k] for k in Considered_Future_Lines[d]) ==
                        quicksum(x[i,j] for j in active_connections[d]) + quicksum(HAFL[
                        active_aircraft[i]][p] for p in frozen_connections[d]) + quicksum(q[i
                        ,k] for k in future_connections[d]))

    m.optimize()
    #Save routing solution and prepare for next day iteration
    line_solution = m.getAttr('x', x)
    Current_Routing_Solution = [(i,j) for (i,j) in line_solution if line_solution[i,j] > 0.9]
         #this routing will be frozen
    task_solution = m.getAttr('x', y)
    Maintenance_Planning_Solution = [(i,t,j) for (i,t,j) in task_solution if task_solution[i,
        t,j] > 0.9]
    future_line_solution = m.getAttr('x', q)
    Future_Routing_Solution = [(i,k) for (i,k) in future_line_solution if
        future_line_solution[i,k] > 0.9]

    #i need to update: frozen_lines, Flight_hours_recorded, constructed_route, HAFL and
        starting_aircraft list
    for i,j in Current_Routing_Solution:
        frozen_lines.append(j) #update the frozen lines
        HAFL[starting_aircraft[i]][j] = 1 #update HAFL
        Constructed_Routes[starting_aircraft[i]].append(j) #update the constructed routes
        Flight_Hours_Recorded[starting_aircraft[i]] += Lines[j]['Flight Hours'] #update the
            number of FH each AC has recorded

    for i in all_aircraft:
        Most_Recent_Maint_Schedule[i] = []    #reset the maint schedule of the active AC so
            that we can fill it in again

    for i,t,j in Maintenance_Planning_Solution:
        Most_Recent_Maint_Schedule[all_aircraft[i]].append([t,j]) #rebuild the updated
            maintenance schedule

    #Now I need to specify how many flight hours were flown after each line in the frozen
        lines, for future planning needs
    for i in TailNumbers: #for every AC
        for m in range(len(Constructed_Routes[i])): #for every line that it has been assigned
             so far
            the_line = Constructed_Routes[i][m] #identify the line
            flown_lines_before_it = Constructed_Routes[i][:(m+1)] #identify which lines are
                assigned to the AC before the considered line
            Flown_FH_after_line[the_line] = sum([Lines[b]['Flight Hours'] for b in
                flown_lines_before_it]) #determine how many FH are flown before the line
                starts

    #Keep track of the current scores of the routes
    for i in all_aircraft:
        MaintScore[i]=0 #reset the score for the active AC cause we are going to recalculate
            them
        for t,j in Most_Recent_Maint_Schedule[i]: #iterate through the planned maintenance
            for AC i
            Score_days = (AllTaskData[i][t]['Task Due Date'] - (Lines[j]['EndDate']+1))*
                average_FH_per_day #Lost flying hours due to task being executed before its
                due date
            Score_FH = AllTaskData[i][t]['Task FH Due'] - Flown_FH_after_line[j] #lost flying
                 hours due to task being executed before remaining legal FH
            overall_score = AllTaskData[i][t]['TaskMH']*min(Score_days,Score_FH)
            MaintScore[i] += overall_score

    #Now to determine the new active aircraft for the next iteration
    starting_aircraft = []
    for it in Constructed_Routes:
        next_active_day = Lines[Constructed_Routes[it][-1]]['EndDate'] + 1
        if next_active_day == day + 1:
            starting_aircraft.append(it)
    #the transit AC for the following iterations
    transit_aircraft = []
    for it in Constructed_Routes:
        if Lines[Constructed_Routes[it][-1]]['EndDate'] >= day+1 and Lines[Constructed_Routes
            [it][-1]]['EndDate'] < day+1+forecasting_period:
            transit_aircraft.append(it)
    active_aircraft = starting_aircraft + transit_aircraft
    non_starting_aircraft = []
    for i in TailNumbers:
        if i not in starting_aircraft:
            non_starting_aircraft.append(i)
    all_aircraft = starting_aircraft + non_starting_aircraft
    #determine the tasks that the non-active AC have planned and will need to replan next
```

```
        iteration
    for i in non_starting_aircraft:
        non_starting_task_set[i] = [t for (t,j) in Most_Recent_Maint_Schedule[i]]

#Print the solution scores
print('Final Scores: '+str(MaintScore))
print('Final Total Score: '+str(sum(MaintScore.values())))
stopwatch_end = time.time()-stopwatch_start
print('Total Elapsed time: '+str(stopwatch_end))
```