Aircraft Preventive Maintenance: Task Packaging & Scheduling

Author: Dino Spirtovic

Supervisor: Dr. Rene Bekker

Supervisors: Drs. Gerald Poppinga
Dr. Arjan de Jong

August 29, 2018
v1.2
Abstract

Aircraft maintenance takes up a major part of the airlines operational expenses and strongly influences aircraft availability. The purpose of this study is to identify or develop methods to efficiently package and schedule preventive maintenance tasks for fleet of aircraft. Because of the large number of maintenance tasks that need to be scheduled for each aircraft, we aim to reduce the complexity of the scheduling problem by packaging maintenance task into equalized working packages prior to scheduling. Therefore, our research can be roughly divided into three parts. The first part focuses on identifying or developing a suitable method that can be used to cluster maintenance tasks with similar planning intervals. Because we failed to obtain desirable clusters using popular clustering algorithm from the field of machine learning, we developed a new clustering method and show that it outperforms the other considered algorithms for our problem. The second part of our study aims to efficiently split the clustered tasks into equalized working packages such that they can be scheduled during aircraft ground times. We proposed a mixed integer linear programming formulation of the problem and solved it to prove that it is able to obtain optimal packages for our objective and capacity restrictions. Finally, we investigate which methods are capable of producing efficient maintenance schedules for fleet of aircraft given their flight plan and available maintenance opportunities such that the number of maintenance visits is minimized. Using a test case, we show that our heuristic scheduling algorithm is able to produce efficient maintenance schedules with little computational power.
## Contents

1 Introduction ................................................................. 7  
1.1 Problem background ....................................................... 7  
1.2 Research goal and approach ........................................... 8  
1.3 Host organization ......................................................... 9  
1.4 Thesis outline ............................................................ 9  

2 Aircraft Maintenance Programs .......................................... 11  
2.1 Maintenance task development ....................................... 11  
2.2 Maintenance planning document ..................................... 11  
2.3 Maintenance checks ..................................................... 12  
2.3.1 Block checks ......................................................... 13  
2.3.2 Phased checks ....................................................... 13  
2.4 Maintenance scheduling ............................................... 13  

3 Literature Review .......................................................... 15  
3.1 Related work ............................................................. 15  

4 Maintenance Task Clustering .............................................. 17  
4.1 Methods ................................................................. 17  
4.1.1 Dissimilarity measures ............................................. 17  
4.1.2 Clustering algorithms .............................................. 17  
4.1.3 Sort&Cut algorithm ................................................ 19  
4.1.4 Evaluation criteria ................................................ 20  
4.2 Interval data ............................................................. 20  
4.2.1 Data exploration ................................................... 21  
4.2.2 Data preparation .................................................... 22  
4.3 Results ................................................................. 24  
4.3.1 Experimental setup ............................................... 24  
4.3.2 Imputation influence .............................................. 24  
4.3.3 Algorithm performance .......................................... 26  
4.3.4 Parameter sensitivity ............................................. 26  
4.3.5 Clustering non-blank subsets .................................... 27  
4.3.6 Sort & Cut ......................................................... 28  
4.4 Conclusion .............................................................. 29  

5 Maintenance Task Packaging ............................................ 31  
5.1 Methods ................................................................. 31  
5.1.1 Bin-packing problem .............................................. 31  
5.1.2 Task-packing problem formulation ............................... 31  
5.1.3 Implementation .................................................... 34  
5.2 Workload data .......................................................... 35  
5.2.1 Data exploration ................................................... 35  

5
### 5.2.2 Data preparation

5.3 Results

5.3.1 Experimental setup

5.3.2 Performance

5.3.3 Run-time

5.4 Conclusion

### 6 Maintenance Scheduling

6.1 Methods

6.1.1 Exact solution

6.1.2 Heuristic solution

6.1.3 Comparison

6.2 Task data

6.2.1 Data preparation

6.2.2 Generated data

6.3 Results

6.3.1 Experimental Setup

6.3.2 Single aircraft scheduling

6.3.3 Fleet scheduling

6.3.4 Run-time

6.4 Conclusion

6.4.1 Future work
Chapter 1

Introduction

1.1 Problem background

Fierce competition and strict safety regulations have forced airlines to manage their operations as efficiently as possible. The demand for efficiency has inspired many studies in the field of Applied Mathematics and Operations Research to address a wide variety of airline operations problems, leading to the development and refinement of various optimization models and algorithms. This has helped the civil aviation industry in becoming one of the most cost-efficient industries in the world.

Maintenance task packaging and planning are hot topics in the aviation industry and for a good reason. Namely, aircraft maintenance takes up a major part of the airlines operational expenses and strongly influences aircraft availability. The airplane manufacturer provides the Maintenance Planning Document (MPD) containing a list of preventive maintenance tasks and intervals at which the tasks must be performed for continued airworthiness of the aircraft. Furthermore, a pre-approved generic maintenance program is provided. This program consists of tasks packed in so called letter checks, including a generic schedule for these checks. However, the lifetime of many tasks is not utilized efficiently in the generic program.

Using the MPD as a basis, aircraft operators may also develop their own custom maintenance programs that better suits their flight operations and planning strategy. If managed effectively, a custom program may lead to a optimal interval utilization and a reduction in maintenance costs by reducing aircraft downtime. Additionally, the increased availability of the aircraft allows for more revenue to be generated. Nevertheless, constructing a efficient custom maintenance program remains a complex tasks. This is why some operators still use the generic maintenance program provided by the aircraft manufacturer.

As mentioned previously, NLRs mission is to increase sustainability, safety and efficiency of transport. Therefore, the NLR is developing a software suite that can assist airlines in establishing customized maintenance programs. This research will focus on the development of methodology for maintenance task packaging and scheduling which is crucial for the optimization module of the software suite.
1.2 Research goal and approach

The purpose of this study is to identify and develop task packaging and scheduling methods that are suitable for the optimization module of NLRs maintenance software. More specifically, we aim to answer the following main research question:

- How can aircraft maintenance task packaging and/or scheduling be optimized?

Maintenance planning on fleet level is a complex problem because of the large number of individual tasks and the variety of task intervals. Therefore, our initial approach is to reduce the size of the scheduling problem by identifying tasks with similar intervals and group them into maintenance packages. This way we only have to schedule packages instead of each task individually. Moreover, task packaging prior to scheduling has several practical advantages. Namely the complexity of the maintenance program is reduced, which is beneficial for the maintenance staff as well as the regulatory authorities. Furthermore, the operational process of carrying out specific maintenance tasks in fixed packages allows for optimization of those processes at ground level.

Prior to task packaging, we investigate the possibility to cluster maintenance tasks based on their interval using various clustering techniques from the field of machine learning. Clustering on intervals is done to minimize the loss of lifetime for each maintenance task when creating packages. Note that the interval at which a package has to be scheduled depends on the task with the shortest interval inside that package. The next step in our research is to determine how to divide each cluster into efficient packages with balanced workloads.

Additionally, one can save time by performing certain maintenance tasks together during a single maintenance opportunity. To maximize the time saving during the packaging procedure, we propose a mixed integer linear programming formulation (MILP) of the problem that can be solved to obtain the optimal solution. Finally, we examine an MILP and heuristic approach for task scheduling. To illustrate the scalability of the purposed scheduling methods, we first apply them to schedule maintenance for a single aircraft and attempt to extend those methods to work for a fleet of aircraft.

At each of the research steps discussed in the previous paragraph, we compare advantages and disadvantages of the different methods. Thus, we aim to answer the following additional research questions:

- Can tasks be clustered efficiently based on their planning intervals?
- How much access synergy can be obtained by packaging tasks prior to scheduling?
- Can an optimal maintenance schedule be obtained using the proposed methods?
- What are the (dis)advantages of the proposed scheduling methods?

The scope of our research is limited to a single MPD. Nevertheless, our aim is to develop a robust approach to maintenance task packaging and scheduling because an operator might have a wide variety of aircraft models in its fleet. This means we will only use that is common in MPDs for various manufacturers.
1.3 Host organization

Netherlands Aerospace Center (NLR) is the Dutch knowledge institute that is dedicated to identifying, developing and applying advanced technological knowledge in the aerospace sector. Approximately 630 employees with knowledge from various fields, ranging from psychology to mathematics and from aerospace engineering to logistics, are dedicated to innovation. The organization is divided into fifteen departments which are grouped under the following divisions: Aerospace Systems (AS), Aerospace Operations and Aerospace Vehicles. This research is conducted to support a maintenance engineering project at the Aerospace Electronics & Qualification (ASEQ) department, which is part of the AS division.

The organization was founded in 1919 as the Government Service for Aeronautical Studies (RSL) to increase safety for military aviation. As a result of rapid advances in civil aviation at that time, the RSL started focusing on civil aviation as well. Sixteen years later the RSL was cut-off from the government and turned into a foundation, namely NLR. To this day, the company remains non-profit, market-oriented and its activities still relevant to society.

NLRs strategic ambition is to remain the Dutch governments partner of choice for aerospace related matters and keep increasing the competitive strength of national businesses. Nevertheless, the company supports civilian and military clients in the development and deployment of a wide variety of innovative products, not just in the Netherlands but all over the world. Their work also supports the industry and government in establishing safety and environmental policies for aerial transport. Additionally, NLR aims to improve other sectors of transports by applying knowledge and solutions derived from the aerospace sector. These core activities are expressed in the following mission statement:

NLRs mission is to increase the sustainability, safety and efficiency of transport.

The mission is pursued by means of collaboration with various universities and research institutes such as the German Aerospace Center. Furthermore, NLR manages specialized research facilities, e.g. flight simulators, high-speed wind tunnels and a durability testing laboratory. This enables NLR to provide their clients with objective and independent analysis.

1.4 Thesis outline

We begin by providing the reader with basics of aircraft maintenance programs and relevant terminology necessary for a better understanding of the rest of this thesis in chapter 2. Secondly, a review of related work is given in chapter 3. The three main topics of this study are covered in three major chapters, namely maintenance task clustering, packaging and scheduling. In chapter 4 we explore and compare various approaches that can be used to identify tasks with similar planning intervals. In chapter 5 we propose a model to optimally divide tasks with similar intervals into equalized working packages. In chapter 6 we compare two methods that can be used to construct a maintenance schedule for a fleet of aircraft given their flight operation. Note that each of these chapters consists of a short introduction, a methods section, results and conclusion. Finally, we provide some suggestions for future research to extend and improve the methods developed during this study.
Chapter 2

Aircraft Maintenance Programs

Just like with any other mechanical equipment, timely maintenance is necessary to keep it operational. For safety purposes, the operational conditions of airplanes are usually regulated by airworthiness authorities. To keep an airplane in airworthy condition and to minimize the risk of component or system failure, the operator is obligated to have an approved maintenance program. The operator can use a pre-approved generic program or construct a customized program. The generic program is prepared by the aircraft manufacturer and consists of tasks packed in so called letter checks, including a generic schedule for these checks. The generic program is designed such that it can be used by most operators. Normally, the (generic) letter checks packaging does not utilize the task intervals efficiently because it does not consider the precise usage of the aircraft, which is related to each airlines unique operations. The same can be noted for the generic schedule that will rarely fit an airlines unique operational strategy. Constructing a customized program requires more effort but can lead to large economic advantages if done properly.

2.1 Maintenance task development

The routine or preventive maintenance task development for aircraft is done according to the Maintenance Steering Group method (MSG-3) and is the only method accepted by the airworthiness authorities for commercial airplanes today [Ackert, 2010]. The MSG-3 method identifies appropriate routine maintenance tasks that serve as countermeasures for specific functional failures. The intervals at which these tasks should be performed are determined using historical data of earlier or similar aircraft models and performance data of certain parts provided by the part manufacturers. An airplane manufacturer prepares the initial minimum routine maintenance requirements, in accordance with MSG-3 methodology, and submits it in a Maintenance Review Board (MRB) report to the regulatory authorities. The maintenance and inspection requirements defined in the MRB report are used to develop an approved MPD. The approved MPD contains a list of all mandatory preventive maintenance tasks for every configuration of the aircraft model, including additional inspection tasks defined in the so called Certification Maintenance Requirements and Airworthiness Limitations documents. Thus, the MPD contains all the information that is required to develop a customized maintenance program.

2.2 Maintenance planning document

Each tasks listed in a MPD contains the information illustrated in table 2.1. Columns ‘access’ and ‘zone’ indicate which panels have to be removed to perform the task and which aircraft zones
Table 2.1: Task list header from MPD including a example task

are relevant for the task respectively. It is important to note that ‘access’, ‘men’, ‘task M.H’ and ‘access M.H’ are listed per zone. A illustrative example of the aircraft zones is given in figure 2.1. Furthermore, the ‘access’ column is empty if no panels need to be removed. For example, a task that can be performed without having to remove panels is a visual check of the airplane hull. The task type to is indicated in the ‘type’ column and can vary from simple visual checks to overhauls of major components. The ‘skill’ column indicates what type of skill the maintenance engineer requires to perform the task, e.g. electrical for task involving electricity generation or distribution. As mentioned previously, the ‘interval’ column contains a time at which the tasks need to be performed in flight cycles, flight time, calendar time or any combination of these three units. Furthermore, ‘threshold’ indicates the time at which the task has to be performed for the first time after the aircraft goes into service (from new state) and ‘interval’ indicates the time at which it needs to be repeated after the threshold is reached. In the example the interval is noted as C which is equivalent to 450 flight cycles, 6000 flight hours or 20 months. We will discuss the ‘threshold’ and ‘interval’ column in more detail in chapter 4. The ‘men’ column indicates how many engineers are required for the task while ‘task M.H’, ‘access M.H’ and ‘prep M.H’ specify how much man hours are needed to complete the actual maintenance, remove panels and prepare for the task respectively. Note that the access M.H is empty if no panels need to be removed, that is when the ‘access’ column is empty as well. The last column specifies to what configuration or modification of the aircraft model and engine the task is applicable to.

Figure 2.1: Zone indication for both sides as shown in the A320 maintenance manual

2.3 Maintenance checks

As mentioned in the previous section, preventive maintenance tasks can vary from short service checks with daily/weekly intervals to overhauls of major components with an interval of more than 10 years. The minor inspection/maintenance tasks, that do not require specialized equipment or personnel, can be performed at line. This means that some (line maintenance) tasks can be done while the airplane is parked near a gate or terminal. Tasks that require a longer period of time, more personnel and the use of non-standard equipment need to be performed at the base. This is also known as base maintenance. A common approach in the industry for the development of maintenance programs is to group tasks into maintenance packages or checks with similar intervals.
prior to scheduling. There are two common packaging methods, namely the block- and phased packaging. These two methods are illustrated in figure 2.2 and briefly described in the subsections below.

### 2.3.1 Block checks

Tasks with similar intervals are grouped into large ‘blocks’ and assigned an interval that is not longer than the shortest task interval within the block. Based on the length of this interval, a letter is assigned to the block or package. This is why block checks are commonly known as letter checks. Blocks with short intervals are assigned the letter A. Medium sized (in terms of required effort) blocks with monthly to (bi-)yearly intervals are denoted by the letter C. Maintenance blocks that require a lot of resources, usual with intervals larger than 10 years, are called D-Checks.

Packaging tasks into block checks simplifies the scheduling work. However, lifetime of certain tasks is lost using this method. Remember that the interval of the block cannot be longer than the shortest task interval within the block, otherwise a task might not meet its maintenance deadline. Other disadvantages include a long consecutive downtime during maintenance and uneven requirement of resources over time.

![Figure 2.2: Block- and phased check approaches illustration in figures a and b respectively.](image)

### 2.3.2 Phased checks

The phased check method involves splitting the block checks into smaller packages and scheduling them evenly before their deadline. This method aims to gain economic advantages by increasing airplane availability and balancing the workloads over time. To maximize these advantages more effort is necessary with regards to task packaging & scheduling.

### 2.4 Maintenance scheduling

An important regulatory rule for task scheduling is that tasks must be scheduled before their 100% interval or deadline. Additionally, there is no loss in lifetime if tasks are performed within their 80-100% interval. Performing a task before its 80% interval leads to a large loss in lifetime, namely the full difference between its 100% deadline and the time at which it is performed. This can be observed in the illustrative example of this scheduling rule in figure 2.3. The boxes visualize the 80-100% interval for both cases. Observe that the next deadline is 120% of the interval time away if the previous maintenance took place at 80%. And if one performs maintenance at 79% of its
interval, the next deadline is 100% of the interval time away. That is a total loss of 21% for the interval lifetime.

Figure 2.3: Scheduling rule illustrated
Chapter 3

Literature Review

3.1 Related work

Most literature on aircraft maintenance scheduling focuses on fleet assignment and routing. [Gavranis and Kozanidis, 2015] propose an exact solution method to maximize fleet availability by deciding which aircraft to assign to each flight while meeting certain maintenance requirements. Their model focuses on a military aircraft for which some flight and maintenance time needs to be met to keep it operational. Important to note is that their model does not consider the planning of distinguishable tasks but simply assume a maintenance time requirement for each period. Similarly, [Al-Thani et al., 2016] propose a heuristic method to construct feasible aircraft routes that cover all flights and satisfy some number of maintenance visits. The objective of the model proposed by [Sriram and Haghani, 2003] aims to minimize maintenance costs for a fleet of commercial aircraft by assigning aircraft to scheduled flights and deciding where maintenance should take place. In contrast to [Gavranis and Kozanidis, 2015] and [Al-Thani et al., 2016], they distinguish 4 different maintenance checks that needs to performed timely.

Research that relates most to ours is that of [Hlzel et al., 2012]. Their general approach to solving the maintenance scheduling problem is similar to ours because their proposed optimization method for maintenance scheduling also includes re-packaging generic letter checks into smaller packages that can be scheduled and performed during nightly aircraft ground times. Furthermore, their planning optimizer also simulates aircraft utilization to determine when and at what opportunity maintenance should be performed. The difference with our approach is that they schedule tasks based on a predicted urgency to demonstrate the application of their scheduling algorithm for future aircraft systems where maintenance is prognosis-based. Another major difference is that a large part of our study focuses on identifying/developing methods that can be used to obtain clusters of tasks with similar intervals and dividing these cluster into equalized packages such that the access costs are minimized.
Chapter 4

Maintenance Task Clustering

The goal of maintenance task clustering is to identify which tasks can be grouped together into maintenance checks/packages with a minimal loss of lifetime. Thus, in this chapter we instigate how clustering algorithms from the field of machine learning can be used to match tasks with similar intervals. Our initial approach is to create some ‘baseline’ clusters by considering each unique combination of the interval variables as a cluster. Secondly, various clustering algorithms are tested in combination with various data preprocessing methods to identify which method can be used to automatize the task clustering process. In addition to the classical clustering algorithms, a new clustering algorithm is designed and tested.

4.1 Methods

4.1.1 Dissimilarity measures

Every clustering algorithm is driven by a metric that expresses the (diss)similarity between data objects. Therefore, the choice of a metric that expresses the (dis)similarity accurately is crucial to obtain desirable clusters. Finding the appropriate metric is often an iterative process. However, knowledge of what the data represents and its context can contribute to finding an appropriate metric. There are various generic functions that can be applied to a wide-variety of problems. The most commonly applied distance measure is the euclidean distance. The distance is defined as the sum of squared differences between each variable of two objects.

4.1.2 Clustering algorithms

In this subsection we give a brief overview of the considered clustering algorithms. For more details we refer the reader to [Alpaydin, 2010] and [Kriegel et al., 2011].

Centroid-based

Centroid clustering algorithms require a pre-defined number of clusters \( k \) as input parameter. The algorithm initializes \( k \) centroids in the range of the data objects. Based on the distance measure, objects are assigned to the nearest centroid. Secondly, the centroids are moved towards the center of the objects belonging to them. In the case of the \( k \)-means algorithm the centroids are moved to the average location (center) of the cluster. The \( k \)-medoids algorithm moves the centroid to the object closest to the average location. The steps of determining and assigning the
nearest neighboring data objects to each centroid and moving the centroid towards the center of that neighborhood is repeated until the movement of all the centroids converges. Note that the resulting cluster are strongly influenced by the initial position of the centroids.

Hierarchical

Hierarchical clustering algorithms split (or merge) objects to create a hierarchical structure for every object known as a dendrogram. A dendrogram indicates at which distances the clusters/objects are merged or split. The agglomeration (Wards) algorithm initializes all individual objects as clusters and iteratively calculates the distances and merges clusters with the shortest distance to each other. This is done until there is only one large cluster left. Hierarchical clustering can also be done from top to bottom, starting with one large cluster and successively splitting the cluster. This is known as the divisive approach. Note that using both approaches, clusters which are most similar to each other are lower in the hierarchy. The constructed dendrogram can be cut at a certain height $h$ to obtain the corresponding number of clusters such that the clustered objects are at most $h$ apart from each other. Therefore, the algorithm does not need to be run again to change the number of resulting clusters.

Distribution-based

Distribution-based clustering identifies clusters by fitting multiple distributions to the data. The expectation-maximization algorithm for example, attempts to fit $k$ Gaussian distributions. The algorithm is similar to the centroid-based algorithms. It starts by initializing $k$ Gaussian distributions with random parameters, after which it calculates the error between the data points and fitted distribution and updates the parameters of the Gaussian distribution by maximizing the probabilities that each point within a cluster belongs to that distribution. This is repeated until all the model parameters converge. The key difference between the expectation-maximization (EM) algorithm and centroid-based methods is that a standard deviation is included in the model. This enables the cluster shape to take the form of an ellipse instead of just a simple circle.

Density-based

Density-Based Spatial Clustering of Applications with Noise (DBSCAN) is a popular algorithm which identifies clusters of various shapes by iteratively grouping together data points that are located in a ‘neighborhood’ and marking data points as noise if they are not. The algorithm has two parameters, namely $\epsilon$ and $\text{minPt}$. The first parameter indicates the range around a certain point that should be considered a neighborhood and the latter specifies a minimum number of points required to form a cluster. The algorithm starts at a random data point and determines if there are enough neighboring points based on the given parameters. If there are not enough neighboring points, the selected point is marked as noise and a new unvisited data point is selected at random. If there are enough points, the selected point and all its neighbors are marked as a new cluster. The algorithm repeats the neighborhood search for each unvisited point in the new cluster and extends the cluster to all points within the $\epsilon$ range. When there are no more unmarked points within the clusters range, the algorithm chooses a unvisited point at random again in a attempt to find more clusters or identify noise. The search process is terminated when each data point has been visited.

Note that the choice of $\epsilon$ is very important for this algorithm. A too large value for $\epsilon$ might lead to two clusters being merged directly or via a noisy point. However, choosing the parameter too low will lead to many points being falsely marked as noisy. To overcome this problem, an extended version of DBSCAN exists, called HDBSCAN. HDBSCAN only requires the $\text{minPt}$ as input. The algorithm start with a random data point and expands a area $\epsilon$ around it until at least
minPt points are in the neighborhood. Formally, the ϵ is called the core distance in the context of HDBSCAN. Secondly, the mutual-reachability distances between each data point is calculated. Given the distance between two points $d_{ij}$ and the core distances of these points, i.e. $c_i$ and $c_j$, the mutual-reachability is given by $\max d_{ij}, c_i, c_j$. Using the mutual-reachability distance, a minimum spanning tree is constructed and translated into a dendrogram. Finally, the dendrogram is cut such that it returns clusters of minimum size ($\text{minPt}$).

### 4.1.3 Sort&Cut algorithm

For the purpose of task clustering we developed a clustering algorithm, named Sort&Cut, which splits any multi-column data set recursively such that the difference between any column variable is at most $l$ for any resulting cluster. The pseudo-code is provided below, under algorithm 1. The function begins sorting $M$ by the values of the first column from low to high. Using the lagged difference and the sorted values, a vector $v$ is produced which contains the cumulative sum of the differences between the sorted values of the first column. This vector can be used to determine where the data should be split such that the values of the first column are no more then a factor $l$ apart from each other for the upper part of the split. Two new instances of the function are called, each with one part of the split data. The upper part of the cut repeats the procedure for the next column variable. This process is repeated for each upper split of the recursion until all columns have been visited. The clusters are then labeled and returned. The lower part of the split continues to shrink until the range of the values is within the limit. Then it moves on to the next column.

**Algorithm 1** Sort&Cut clustering

```plaintext
procedure sort_and_cut(M, d, l, n)
    m ← num_columns(M)
    if n < 2 then
        label(M) ← n
        return M, n + 1
    end if
    M ← sort_increasing(M, by dimension d)
    v ← lagged_diff(M(d))
    v ← v / M(d)
    v ← cumulative_sum(v)
    if max(v) <= l then
        if d < n then
            return sort_and_cut(M, d + 1, l, n)
        else
            label(M) ← n
            return M, n + 1
        end if
    else
        i ← index_first(v > l)
        M(1, i), n ← sort_and_cut(M(1, i), d + 1, l, n)
        M(i + 1, end), n ← sort_and_cut(M(i + 1, end), d, l, n)
        return M, n
    end if
end procedure
```
4.1.4 Evaluation criteria

Clustering algorithms fall under the category of ‘unsupervised learning’ methods. In our case we do not have a response variable which can be used to validate clusters. Therefore, to determine if the clusters are ‘good’ enough the results have to be analyzed manually. This can be done by visualizing the clusters or by using some metric that measures (dis)similarity of objects within clusters. In both cases, contextual knowledge of the clustered data is essential for the evaluation process.

Silhouette coefficient

The silhouette coefficient, as introduced by [Rousseeuw, 1987], ranges from −1 to 1 and measures how well a object \((i)\) fits inside a cluster by comparing the average distance to objects in the same cluster \((a_i)\) with the highest average distance to objects in other clusters. This is formalized in equation 4.1 where \(b_i\) denotes the average distance to objects of most distant cluster. A value close to 1 indicates a good fit. The coefficients of the objects within a cluster can be averaged to determine the (average) quality of the clusters. An overall quality can be expressed by averaging the cluster averages.

\[
s_i = \frac{b_i - a_i}{\max \{a_i, b_i\}}
\]

Relative cluster width

The relative cluster width \(w_i\) for an arbitrary cluster \(i\) containing a vector of tasks \(v_i\) can be calculated using equation 4.2. This quality metric is interesting specifically for our problem. Reason being that if the relative cluster width is too large one would throw away a lot of lifetime if the clustered tasks are grouped in working packages.

\[
w_i = \frac{\max (v_i) - \min (v_i)}{\max (v_i)}
\]

4.2 Interval data

The threshold and interval data is extracted from a MPD and structured into a list with three variables, namely: flight hours (FH), flight cycles (FC) and calendar hours (CH). Thus, each task has 6 variables, 3 for the threshold and 3 for the intervals. In the remainder of this paper we will formally refer to these 6 variables as intervals. Some variables may be blank, indicating that the variable does not have to be considered for that task. Important to note is that we ignore intervals that refer to national requirements, vendor recommendations or notes. Reason for this is that this information was not present in the document or unobtainable with NLRs MPD parser at the time this study was conducted.

In the remainder of this section we explore the interval data to point out two main obstacles to interval clustering, namely the large number of blank values and distribution of the non-blank values. Finally, we prepare the data for clustering.
Many maintenance tasks have the same combination of interval values because they originally belong to a generic check. Of the 1732 tasks we extracted from the MPD there are 418 unique combinations of values for the 6 interval variables, including blank values. To identify which tasks have similar intervals one can simply cluster on the unique combinations of these intervals. Table 4.1 contains an overview of the number of tasks and unique combinations for tasks with defined threshold and interval variables. There is one notable insight we can extract from this summary, namely less than half of the tasks have a defined threshold.

Blank values

Figure 4.1 shows that blank values are common because many tasks are based on a single or a combination of two variables. Furthermore, figure 4.1 provides insights into the pattern of the blank values between variables. Again we can see that the values are mostly blank for the threshold variables. One can also observe that the most common combination of blank variables is the one where all 3 threshold variable are blank. Additionally, note that certain combinations have almost no blank values, e.g., threshold FH and threshold FC. This means these variables occur as non-blank together, even though individually they have the most blank values in the set.

Table 4.1: Interval data summary, values are expressed as number of tasks or unique combinations with defined values for the threshold or interval variables

<table>
<thead>
<tr>
<th></th>
<th>Thresholds</th>
<th>Intervals</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tasks</td>
<td>856</td>
<td>1724</td>
<td>1732</td>
</tr>
<tr>
<td>Unique combinations</td>
<td>267</td>
<td>316</td>
<td>418</td>
</tr>
</tbody>
</table>
Distributions

The boxplots in figure 4.2 show the distributions of the non-blank values for each interval variable. The wide interquartile range indicates a high dispersion of the values. Thus, we do not expect to bundle the objects into a few large clusters. Instead, we can foresee that many small clusters are required if we want to obtain structures with a strong structure. Furthermore, figure 4.2 also exposes certain outlier values. These values are indicated as black dots in the plot. These outliers can not be grouped with any other task efficiently and should be scheduled separately. However, we will keep the outliers in our clustering set to investigate how the considered algorithms cope with them.

(a) Flight and calendar hour distribution, horizontal grid lines mark steps of 5 years
(b) Flight cycle distribution

Figure 4.2: Box plots of the non-blank values

4.2.2 Data preparation

As noted previously, there are various combinations of blank and non-blank variables. Thus, there are two approaches one can take to deal with these values, namely:

1. Cluster on all 6 variables by imputing the blank values.

2. Subdivide the data such that each subset of interval variables only contain non-blank values.

For this research we explore both approaches. However we limit the exploration for the second approach to a few representative subsets from which we can draw some general insights.

Split

For the first approach we will split the data into two sets, namely one where all the threshold variables are blank and the other where at least one is available. Considering these sets separately is beneficial for the blank value imputation method. Imputing 5 out of 6 variables based on a single value is a inaccurate way to fill the blank values. Thus, the nearest neighbor (NN) imputation will work better if the sets are considered separately during clustering.
For the second approach we will sample a few subset combinations of intervals for which there are no blank values. The considered subsets are visualized in figure 4.3. To investigate if the set size or dimension influences clustering algorithm performance, subsets of different sizes and dimensions are sampled, namely:

- All variables available (Set size: 28)
- Interval FC and threshold FC (Set size: 130)
- Interval FC, interval FH, threshold FC and threshold FH (Set size: 162)
- Interval FH (Set size: 27)

Figure 4.3: Frequency plot showing the number of blank interval values and their pattern among variables for unique combinations

(a) Blank values imputed with 0  
(b) Blank values moved to nearest neighbor

Figure 4.4: Interval CH, FH & FC after imputation
Blank value imputation

The blank values need to be imputed with some numeric values in order to define a distance or similarity between objects. Just like the choice of the similarity metric, the value imputation method will influence the clustering results drastically. This means we need to impute the values in a way such that it leads to desirable clustering results.

Two approaches are proposed and tested for this study. The first approach is to set all the missing values to a number outside the range of the available values, e.g. 0. This way the blank values would be equal to each other but distinct from the available values. The second approach is to find the nearest neighbor of an object using its non-blank values and filling its blank values with those of the nearest neighbor. The results of both imputation methods are visualized, for the interval variables, in figure 4.4.

Scaling

Algorithms such as (H)DBSCAN are known to perform better when the data is scaled. A common scaling technique is standardization, which scales a value $x$ by subtracting the variables mean ($\mu$) from it and dividing that by the variables standard deviation ($\sigma$). Thus, scaled values can be interpreted as a deviation from average value. Another common method used during this study is normalization, where values are normalized between 0 and 1.

4.3 Results

4.3.1 Experimental setup

Cluster analysis is an iterative process. Various data pre-processing methods, algorithms and algorithm parameters were explored during this study. The explored algorithms and data preprocessing methods are summarized in tables 4.2 and 4.3 respectively.

<table>
<thead>
<tr>
<th>Parameter(s)</th>
<th>Methods</th>
<th>Imputation</th>
<th>Scaling</th>
<th>Distance</th>
</tr>
</thead>
<tbody>
<tr>
<td>k-means $k = 10\text{–}50$</td>
<td></td>
<td>zero</td>
<td>standardize</td>
<td>Euclidean</td>
</tr>
<tr>
<td>Ward $h = 0.01\text{–}1$</td>
<td></td>
<td>NN</td>
<td></td>
<td></td>
</tr>
<tr>
<td>EM $k = 10\text{–}50$</td>
<td></td>
<td></td>
<td>normalize</td>
<td></td>
</tr>
<tr>
<td>DBSCAN $\epsilon = 0.01\text{–}0.5$</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\text{minPt} = 2\text{–}10$</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HDBSCAN $\text{minPt} = 2\text{–}10$</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 4.2: Algorithms & parameters

<table>
<thead>
<tr>
<th>Methods</th>
<th>Imputation</th>
<th>Scaling</th>
<th>Distance</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>zero</td>
<td>standardize</td>
<td>Euclidean</td>
</tr>
<tr>
<td></td>
<td>NN</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 4.3: Pre-processing

4.3.2 Imputation influence

The two suggested imputation methods have distinct effects on the resulting clusters. Imputing the blank values with zeros results in clusters where variables are strictly matched based on their values. This means that algorithms tend to cluster combinations that have one or more blank variables in common, which is to be expected because their distance to one another is 0 for the imputed variables and large for the tasks with non-blank values. Two example clusters are sampled from the most prominent results and presented in table 4.4. The table contains 3 clusters sampled
from the results demonstrate the quality of the clusters. Most obtained clusters are strong, e.g.
cluster 5 and 9. However, the algorithm tends to cluster intervals with low values poorly. The
absolute differences between the interval values in cluster 1 are low but their relative difference
are high, which makes that specific cluster weak in practice. Nevertheless, these sequences do not
occur often in the interval data set and can be corrected by cutting the cluster as illustrated by
the dashed line in table 4.4.

<table>
<thead>
<tr>
<th>Cluster</th>
<th>Threshold</th>
<th>Interval</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>CH</td>
<td>FH</td>
</tr>
<tr>
<td>9</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>7400</td>
</tr>
<tr>
<td>9</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>9</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>87600</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>87600</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>720</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 4.4: Sample clusters obtained with DBSCAN(ε = 0.1, minPt = 2) after imputing blank
values with zero’s.

<table>
<thead>
<tr>
<th>Cluster</th>
<th>Threshold</th>
<th>Interval</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>CH</td>
<td>FH</td>
</tr>
<tr>
<td>28</td>
<td>41700</td>
<td>24000</td>
</tr>
<tr>
<td>24000</td>
<td>8300</td>
<td>5100</td>
</tr>
<tr>
<td>28</td>
<td>42000</td>
<td>24000</td>
</tr>
<tr>
<td>28</td>
<td>42000</td>
<td>24000</td>
</tr>
<tr>
<td>28</td>
<td>40600</td>
<td>24000</td>
</tr>
<tr>
<td>28</td>
<td>42000</td>
<td>24000</td>
</tr>
<tr>
<td>2</td>
<td>12000</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>87600</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>87600</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>87600</td>
<td>20500</td>
</tr>
<tr>
<td>2</td>
<td>87600</td>
<td>21000</td>
</tr>
</tbody>
</table>

Table 4.5: Sample clusters obtained with DBSCAN(ε = 0.1, minPt = 2) after imputing blank
values with their NN.

Imputing the blank variables of objects with values of their nearest neighbor naturally leads to
the imputed values being clustered together with their nearest neighbor. The main idea of this
method is to identify combinations that are similar even when one or two variables are disjoint.
The results indicate that this works well when one or two variables are imputed. However, this
imputation method produces some undesirable results. For example, observe cluster 2 in table
4.5 where the values are matched correctly when we observe each variable individually, yet these
combinations would not match efficiently in practice because the non-blank values are disjoint.
Thus, NN imputation is not advisable and the remaining results are based on data where the blank
values are imputed by zeros. Finally, both imputations were able to identify the ‘easy’ clusters. Examples can be found in tables 4.4 and 4.5 labeled as clusters 9 and 28 respectively.

4.3.3 Algorithm performance

A general summary of insights obtained during experimentation with various clustering algorithms and their parameters is summarized in table 4.6. In our search for good clusters using the considered algorithm, the advantages of each method are listed under pros and disadvantages under cons.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Pros</th>
<th>Cons</th>
</tr>
</thead>
<tbody>
<tr>
<td>k-means</td>
<td>Difficulty to find the right number of clusters for our problem.</td>
<td>Difficult to find the right number of clusters for our problem.</td>
</tr>
<tr>
<td></td>
<td>Outliers are included in nearest cluster.</td>
<td>Outliers are included in nearest cluster.</td>
</tr>
<tr>
<td>Wards</td>
<td>Subdivides data into clusters with specified maximum distance, i.e. cut height.</td>
<td>Cut height lacks intuition for higher dimension and scaling, thus requires effort to find proper height.</td>
</tr>
<tr>
<td></td>
<td>Outliers are treated as single object clusters.</td>
<td>Outliers are treated as single object clusters.</td>
</tr>
<tr>
<td></td>
<td>Easy parameter tuning.</td>
<td>Easy parameter tuning.</td>
</tr>
<tr>
<td>EM</td>
<td>Max distance ((\epsilon)) for objects to be considered as (part of) a cluster can be specified.</td>
<td>Unable to obtain any good cluster</td>
</tr>
<tr>
<td></td>
<td>Detects outliers.</td>
<td>Detects outliers.</td>
</tr>
<tr>
<td>DBSCAN</td>
<td>Max distance ((\epsilon)) for objects to be considered as (part of) a cluster can be specified.</td>
<td>Finding proper (\epsilon) requires effort.</td>
</tr>
<tr>
<td></td>
<td>Detects outliers.</td>
<td>Points positioned in a sequence will be clustered together, the object with the smallest value in the cluster is far from the object with the largest value, see cluster 1 in table 4.4.</td>
</tr>
<tr>
<td>HBDSCAN</td>
<td>Able to produce some strong clusters.</td>
<td>Strength of resulting clusters varies drastically.</td>
</tr>
<tr>
<td></td>
<td>Clusters sequenced data points just like DBSCAN.</td>
<td>Clusters sequenced data points just like DBSCAN.</td>
</tr>
<tr>
<td></td>
<td>Not tunable.</td>
<td>Not tunable.</td>
</tr>
</tbody>
</table>

Table 4.6: Pros and cons of clustering algorithms applied to interval data

4.3.4 Parameter sensitivity

A parameter sensitivity analysis is presented in table 4.7 for the algorithms that were successful in obtaining strong clusters. HDBSCAN was added to the table to illustrate its shortcoming for our case, namely that it is not tunable. This is in contrast to the epsilon parameter of DBSCAN, which can be tuned to increase the mean silhouette coefficient and mean relative cluster width. As expected, decreasing the epsilon reduces the standard deviation of the silhouette coefficient because only objects that are really close to one another are clustered leading to less clusters (and more points being marked as outliers). However, the standard deviation of the relative width of clusters does not seem to change. Furthermore, the max relative width is also insensitive to the parameter change because DBSCAN tends to cluster sequences of points as long as they are in range to connect with each other. The hierarchical (Wards) clustering method produces similar
results as DBSCAN but has the advantage that it does not group sequences. Note that in our case this is a advantage because it produces clusters with a lower relative width by cutting the dendrogram at a $\epsilon$ distance. This advantage can also be observed from the results presented in table 4.7. Clusters obtained with Wards clustering method have lower relative cluster width values for every statistic. Additionally, there are less points marked as noisy resulting in more and larger clusters.

<table>
<thead>
<tr>
<th>$\epsilon$</th>
<th>minPt</th>
<th>Silhouette</th>
<th>Rel. cluster width</th>
<th>Number of noisy pts.</th>
<th>clusters</th>
</tr>
</thead>
<tbody>
<tr>
<td>HB</td>
<td>2</td>
<td>0.49 0.27</td>
<td>35 18 100</td>
<td>45/357</td>
<td>102</td>
</tr>
<tr>
<td>DB</td>
<td>0.20</td>
<td>0.58 0.29</td>
<td>8 10 60</td>
<td>159/357</td>
<td>52</td>
</tr>
<tr>
<td>DB</td>
<td>0.10</td>
<td>0.76 0.22</td>
<td>5 10 60</td>
<td>259/357</td>
<td>35</td>
</tr>
<tr>
<td>DB</td>
<td>0.05</td>
<td>0.83 0.15</td>
<td>4 10 40</td>
<td>309/357</td>
<td>19</td>
</tr>
<tr>
<td>Ward</td>
<td>0.2</td>
<td>0.68 0.26</td>
<td>5 5 40</td>
<td>178/357</td>
<td>72</td>
</tr>
<tr>
<td>Ward</td>
<td>0.1</td>
<td>0.74 0.24</td>
<td>3 5 26</td>
<td>265/357</td>
<td>39</td>
</tr>
</tbody>
</table>

Table 4.7: Algorithm performance, for zero imputed data set

### 4.3.5 Clustering non-blank subsets

Dividing the data such that each variable is non-blank requires more effort compared to the imputation approach since the clustering and parameter tuning has to be done for each subset separately. However, there were no observable benefits to this method compared to clustering on data where blank values are imputed by zeros. Most clusters obtained by clustering on a non-blank subset are identical to clusters we obtained using zero imputation.

Notable results for some subsets of various dimensions are presented in 4.8 to illustrate the similarities with results from the previous section. For example, looking at the subset where both the interval and threshold of the FC variable are non-blank we can see that the cluster with the highest relative width is 60%. This is the same cluster we discussed earlier, namely cluster 1 from 4.3.

<table>
<thead>
<tr>
<th>Subset</th>
<th>$\epsilon$</th>
<th>minPt</th>
<th>Silhouette</th>
<th>Rel. cluster width (%)</th>
<th>Number of noisy pts.</th>
<th>clusters</th>
</tr>
</thead>
<tbody>
<tr>
<td>All</td>
<td>0.25</td>
<td>2</td>
<td>0.68 0.16</td>
<td>7 2 8</td>
<td>20/28</td>
<td>3</td>
</tr>
<tr>
<td>Both FC</td>
<td>0.10</td>
<td>2</td>
<td>0.83 0.10</td>
<td>7 14 60</td>
<td>81/130</td>
<td>18</td>
</tr>
<tr>
<td>Both FC, FH</td>
<td>0.20</td>
<td>2</td>
<td>0.53 0.37</td>
<td>6 6 30</td>
<td>100/162</td>
<td>17</td>
</tr>
<tr>
<td>Both FC, FH</td>
<td>0.10</td>
<td>2</td>
<td>0.71 0.26</td>
<td>4 4 15</td>
<td>127/162</td>
<td>11</td>
</tr>
<tr>
<td>Both FC, FH</td>
<td>0.05</td>
<td>2</td>
<td>0.84 0.10</td>
<td>2 2 9</td>
<td>144/162</td>
<td>7</td>
</tr>
<tr>
<td>Int. FH</td>
<td>0.10</td>
<td>2</td>
<td>0.74 0.14</td>
<td>31 35 90</td>
<td>7/27</td>
<td>5</td>
</tr>
</tbody>
</table>

Table 4.8: DBSCAN performance, for considered subsets
4.3.6 Sort & Cut

To overcome the shortcoming of traditional clustering methods noted in previously in this section, the sort&cut algorithm is designed and tested. The performance of this algorithm is presented in Table 4.9. The input parameter is intuitive and can be interpreted as the maximum relative width between the smallest and the largest value within each cluster. Hence, the algorithm is easily tunable. It can be observed from the result table that the sort&cut algorithm outperforms classic clustering methods. Namely, the clusters resulting from sort&cut create more stable clusters, i.e the relative differences between clustered objects are lower while there are less objects marked as noisy. Furthermore, because the sort&cut algorithm considers relative differences it can deal with smaller interval values. Table 4.10 shows that it overcomes a the main issue of (H)DBSCAN. Namely, clustering sequenced points together.

<table>
<thead>
<tr>
<th>max width (%)</th>
<th>Rel. cluster width</th>
<th>Number of</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>mean</td>
<td>sd</td>
</tr>
<tr>
<td>Sort&amp;Cut</td>
<td></td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>10</td>
<td>5.2</td>
</tr>
<tr>
<td>10</td>
<td>5.5</td>
<td>2.6</td>
</tr>
<tr>
<td>5</td>
<td>3</td>
<td>1.3</td>
</tr>
<tr>
<td>DB $\epsilon = 0.20$</td>
<td>8</td>
<td>10</td>
</tr>
<tr>
<td>DB $\epsilon = 0.10$</td>
<td>5</td>
<td>10</td>
</tr>
<tr>
<td>Ward $h = 0.2$</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>Ward $h = 0.1$</td>
<td>3</td>
<td>5</td>
</tr>
</tbody>
</table>

Table 4.9: Sort&Cut algorithm performance compared to classic clustering algorithms, blanks imputed with zeros

<table>
<thead>
<tr>
<th>Cluster</th>
<th>Threshold</th>
<th>Interval</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>CH</td>
<td>FH</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>51</td>
<td>0</td>
<td>13300</td>
</tr>
<tr>
<td>51</td>
<td>0</td>
<td>11300</td>
</tr>
<tr>
<td>51</td>
<td>0</td>
<td>13500</td>
</tr>
</tbody>
</table>

Table 4.10: Sample cluster obtained with Sort&Cut for zero imputed data
4.4 Conclusion

Using the zero imputation method lead to desirable results. The suggested method to avoid imputation, is not worth the extra effort since it did not yield any advantages compared to clustering on the entire set with blank values imputed by zeros. Furthermore, the ability of the considered algorithms to identify similar combinations varies drastically. As noted previously, the interval data is highly dispersed and no obvious cluster shapes can be observed. We were only capable of finding strong clusters with algorithms that allow tuning of a distance parameter that influences at what distance neighboring object are clustered. Such algorithms are DBSCAN and Ward’s hierarchical clustering method. Moreover, HDBSCAN was able to produce some strong clusters but also constructed a lot of weak clusters.

Obtaining only clusters with significantly similar interval variables using the considered clustering algorithms proved to be impossible. Clusters are considered significantly similar if the relative cluster widths are less than 20%. This practical requirement inspired the design of a problem specific clustering algorithm which proved to outperform the classic clustering approach.

Note that the clustering step only needs to be performed once for a MPD associated with a family of aircraft. Thus, the need to automatize the clustering process depends on how many of MPDs need to be included in the software suite.
Chapter 5

Maintenance Task Packaging

In this chapter we propose and test a method to split the previously clustered tasks into equalized working packages. More specifically, we formulate an optimization model with the objective of minimizing access costs by bundling tasks with overlapping access locations. For example, consider a case where two tasks require the panels of the left engine to be removed in order to maintain some engine parts. Performing both of these tasks at the same maintenance opportunity will require the panels to be removed only once, instead of twice if these tasks would be performed at different maintenance opportunities. Thus, bundling tasks with a shared access locations into fixed working packages reduces the total access costs. Furthermore, a method to automate the optimization process is provided together with results for a test case.

5.1 Methods

5.1.1 Bin-packing problem

The problem of aircraft maintenance task packaging can be formulated as a special case of the bin-packing problem. The proposed optimization model for maintenance task packaging is derived from the generic bin-packing model. Therefore, a brief description of the bin-packing problem is provided below. For a more detailed description we refer the reader to [Korte and Vygen, 2000a].

The basic variant of the bin-packing problem can be generally described as the problem of packing objects of different volumes into bins with equal capacities in a way such that the number of used bins is minimized. Maintenance tasks with different costs can be regarded as objects with different volumes and the equalized working packages can be regarded as bins of equal capacities. However, recall that tasks can share access costs if they have overlapping access points. In other words, some objects use less bin capacity when they are binned together. The latter characteristic of our problem is shared with a special variant of the bin-packing problem, namely the Virtual Machine (VM) packing problem. As the name suggests, the VM-packing involves allocating VMs onto servers such that the total amount of shareable memory is maximized.

5.1.2 Task-packing problem formulation

Notation

Before we give the mathematical formulation of the problem, relevant variables are defined below.
\[ N := \text{number of zones} \]
\[ M := \text{number of packages} \]
\[ L := \text{number of tasks} \]
\[ S_k := \text{maximum size for package } k \]
\[ \text{expressed in men hours} \]
\[ C_{iz}^a := \text{access workload for task } i \text{ in zone } z \]
\[ \text{expressed in men hours} \]
\[ C_i^g := \text{task (fixed) workload for task } i \]
\[ \text{expressed in men hours} \]
\[ x_{ik} := \text{binary decision variable} \]
\[ x_{ik} = 1 \text{ if task } i \text{ is in package } k \]
\[ y_{zk} := \text{auxiliary decision variable} \]
\[ y_{zk} \text{ is the access workload for zone } z \text{ in package } k \]

Objective

In contrast to the bin-packing where the objective function is to minimize the number of used packages, the objective function for task packaging is defined such that it minimizes the total access workload given a number of packages. Important to note is that the task and preparation costs are fixed. More precisely, the total task and preparation workloads sum up to a fixed value regardless of the task packaging. That is why only the access costs are included in the objective function. According to our model, only the highest access workload for each represented zone within a bin count towards the total workload of a bin. Therefore, the access workload for a given zone \( z \) and package \( k \) is given by \( \max_i \{ C_{iz}^a \cdot x_{ik} \} \). Summing over all \( z \) and \( k \) leads to the total access costs over all bins. The objective of optimally allocating \( L \) tasks into \( M \) packages can therefore be formalized by the following objective function:

\[
\min \sum_{k=1}^{M} \sum_{z=1}^{N} \max_i \{ C_{iz}^a \cdot x_{ik} \}
\]

Important to note is that this objective function is not linear in the decision variables.

Restrictions

Just like for the bin-packing problem, there are two restrictions for our optimization problem. The first restriction is to limit the total workload in each package to be less than or equal to \( S_k \). The second constraint makes sure that each task is packed exactly once. Both restrictions are formalized below.

\[
\sum_{z=1}^{N} \max_i \{ C_{iz}^a \cdot x_{ik} \} + \sum_{i=1}^{L} C_i^g \cdot x_{ik} \leq S_k \quad \forall k
\]

\[
\sum_{k=1}^{M} x_{ik} = 1 \quad \forall i
\]
Model assumptions

A model assumption is that there are no packaging restrictions between tasks. Thus, every task can be packed together with any other task. In practice, however, certain tasks must be performed at the same time while other tasks can not be performed at the same time. This data is provided in the tasks description field which is difficult to parse and thus unobtainable at the time of our study. Once this data is made available it can be formally added to this model in the form of additional restrictions. Consider having a binary matrix $R$ where $R_{ij}$ equals one when task $i$ must be packed together with task $j$. The restrictions which would force certain tasks to be packed together is given in equation 5.1. If one is interested in doing the opposite, namely restricting packaging of certain tasks combinations, one needs to construct $R_{ij}$ such that it equals one when task $i$ may not be packed with task $j$. Then the same restriction can be used with a ‘less then or equal sign’ instead of the ‘greater then or equal sign’.

\[ \sum_{k=1}^{M} x_{ik} \cdot x_{jk} \geq R_{ij} \quad \forall i \neq j \]  

(5.1)

<table>
<thead>
<tr>
<th>Id</th>
<th>Access code</th>
<th>Zone</th>
<th>Men</th>
<th>Task m/h</th>
<th>Access m/h</th>
</tr>
</thead>
<tbody>
<tr>
<td>XMPL-AA-1</td>
<td>191BB</td>
<td>191</td>
<td>4</td>
<td>3.00</td>
<td>0.10</td>
</tr>
<tr>
<td></td>
<td>192BB</td>
<td>192</td>
<td>4</td>
<td>3.00</td>
<td>0.12</td>
</tr>
<tr>
<td>XMPL-BB-2</td>
<td>191KB</td>
<td>191</td>
<td>1</td>
<td>0.50</td>
<td>0.08</td>
</tr>
<tr>
<td></td>
<td>192KB</td>
<td>192</td>
<td>1</td>
<td>0.50</td>
<td>0.10</td>
</tr>
<tr>
<td>XMPL-AB-2</td>
<td>191BB, 191KB</td>
<td>191</td>
<td>1</td>
<td>0.20</td>
<td>0.18</td>
</tr>
<tr>
<td></td>
<td>192KB</td>
<td>192</td>
<td>1</td>
<td>0.20</td>
<td>0.10</td>
</tr>
</tbody>
</table>

Table 5.1: Illustrative example for access/zone modeling

Another assumption is that access points can be modeled by the zone data instead of the access codes. To precisely calculate the access workload for each package, one needs to extract the access costs for each access code from the MPD. Using the extracted data, one can calculate how much of the workload should be discounted from each bin by determining the intersecting access codes within each bin. Unfortunately, the access codes could not be parsed from the MPD at the time of this study. However, there is an observable pattern between the access code, zone and access m/h column. Observe from table 5.1 that the access codes are related to aircraft zones, e.g. access codes 191KB and 192KB relate to zones 191 and 192, respectively. The last task shows that the access m/h of 191BB and 191KB together is indeed the sum of both access codes individually (see first and second task). Thus, we can use the zone column to ‘guess’ the intersection and approximate the workload that should be discounted. Consider the case where the second and last task from the example are binned together, one would have to discount 0.08 because this cost is already accounted for. To account only once for the costs for each access code that is associated with a set of binned tasks, one can take the highest access cost of each zone. Note that if the first and last task are binned together the proposed method will not account for the cost of removing panel 192KB. Reason is because access codes are sometimes disjoint even if the zones are the same. Nevertheless, determining the total access costs for each bin based on the zone demonstrates the general idea which can be used when the access code data is made available.
Linear program

The mathematical formulation of the task-packaging problem can be rewritten into a system of linear equations, also known as a MILP, in the following manner:

\[
\begin{align*}
\min & \quad N \sum_{z=1}^{N} \sum_{k=1}^{M} y_{zk} \\
\text{s.t} & \quad C_{iz}^{\alpha} \cdot x_{ik} - y_{zk} \leq 0 \quad \forall i, z, k \\
& \quad \sum_{z=1}^{N} y_{zk} + \sum_{i=1}^{M} C_{g}^{\beta} x_{ik} - S_{k} \leq 0 \quad \forall k \\
& \quad \sum_{k=1}^{M} x_{ik} = 1 \quad \forall i \\
& \quad y_{zk} \in \mathbb{R}^{+} \quad \forall z, k \\
& \quad x_{ik} \in \{0, 1\} \quad \forall i, k
\end{align*}
\]

If a feasible solution exists, the optimal task packaging with regards to access synergy can be obtained by solving the above MILP. Important to note is that the number of constraints added to make the problem linear equals \(NML\), since restriction (2.3) is for all \(i, z, k\). The total number of decision variables is also increased by \(NM\) because of the auxiliary decision variables. For this study, the MILP was constructed in MATLAB and solved using the built in MILP solver called ‘intlinprog’.

### 5.1.3 Implementation

#### Solver input

(M)ILP solvers commonly require the objective function and restrictions in the form of a coefficient vector \((\vec{c})\) and coefficient matrix \((A)\), respectively. If a feasible solution for \(A\vec{x} \leq \vec{b}\) exists, the solver searches for the optimal one. However, constructing the coefficient matrix of size \((NML + M + L)\) by \((NM + LM)\) is not trivial. In addition, constructing the coefficient vectors and matrices needs to be done for each cluster of tasks separately because they differ in \(N, M\) and \(L\). In this subsection we present a method to automate this process.

Constructing a coefficient vector for the objective function is straightforward. We have two decision variables \(y_{zk}\) and \(x_{ik}\) with lengths \(NM\) and \(LM\), respectively. The coefficients for \(y_{zk}\) in the objective function are all equal to one while the coefficients of \(x_{ik}\) are zero. Therefore, the coefficient vector for the objective function can be constructed by merging a vector of ones (of length \(NM\)) with a zero vector (of length \(LM\)).

\[
\begin{bmatrix}
y_{11} & y_{12} & \cdots & y_{NM} & x_{11} & x_{12} & \cdots & x_{LM}
\end{bmatrix}
\]

(5.8)

Constructing the coefficient matrix is a bit more complicated because each restriction has a different pattern of coefficients. The method to express equations (or restrictions) as coefficients of a vector of decision variables is explained with an illustrative example below.

Consider constructing the coefficient matrix \(A\) for the left-hand side of an equation defined as \(y_{zk} + x_{ik}\) for all \(z \in \{1\}, k \in \{1, 2\}, i \in \{1, 2, 3\}\). The first step is to create column vectors for \(z, k\)
and \( i \), capturing every combination of these variables. The row vectors are created by considering each combination of \( z, k \) and \( i \) without regard for the pattern of the objective vector. The final step is to simply cross reference the row and column combinations of these vectors. In the case of this example, the coefficients are set to 1 where the column and row vectors match. Note that the last step needs to be hard coded for each restriction from 5.2.

\[
\begin{bmatrix}
1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 \\
\end{bmatrix}
\]

\[
\begin{align}
& y_{11} + x_{11} \\
& y_{12} + x_{21} \\
& y_{11} + x_{31} \\
& y_{12} + x_{12} \\
& y_{12} + x_{32} \\
\end{align}
\]

Packaging procedure

A important difference between the bin-packing model and the previously formulated task-packing model is that we do not include a decision variable that determines the usage of an arbitrary bin because we use a different objective function. Instead we approximate the minimum number of bins. If the solver is unable to find a feasible solution for the approximated number of bins, we increase the number until a (optimal) solution is obtained. This way the number of bins used is primarily minimized after which the optimal allocation of tasks is determined with respect to access costs. The choice for this approach was made to reduce the number of decision variables and the size of the coefficient matrix. Alternatively, one can formulate the objective function to minimize the number of packages analogous to the bin-packing model.

Prior to invoking the MILP solver to allocate tasks into bins, an approximation of the number of bins is made. First, the number of ‘outliers’ or tasks that exceed the desired bin capacity is determined. The same number of bins as the number of outliers is allocated and their capacity set equal to the workload of the outlier. This way the solver will be able to obtain a feasible solution by packaging the outliers into these custom bins. Furthermore, the remaining workload is divided by the desired bin capacity to obtain a tight approximation for the number of bins we need to package the remaining tasks.

5.2 Workload data

The model we discussed in the previous section is customized for the available MPD data. Maintenance task costs are specified as number of men and man hours required to complete a task. In this section we give a brief summary of the data used to model the workloads.

5.2.1 Data exploration

Unlike the interval data, the workload data has missing values. Namely, values marked as TBD, i.e. to be determined. A summary of the workload data is presented in table 5.2. We can observe from the table that task and preparation costs have many missing values. The small or no difference in the quartile values indicates that the non-missing values are concentrated around the median, especially for the access and preparation costs. However, the max observed values show us that there are obvious outliers. These outliers are visualized in 5.3. The MEN variable indicates the
number of men that are needed to perform a task. Thus, the WL is determined by multiplying the TASK column with the MEN column.

<table>
<thead>
<tr>
<th></th>
<th>Task</th>
<th>Access</th>
<th>Prep</th>
</tr>
</thead>
<tbody>
<tr>
<td>Min.</td>
<td>0.010</td>
<td>0.000</td>
<td>0.000</td>
</tr>
<tr>
<td>1st Qu.</td>
<td>0.100</td>
<td>0.000</td>
<td>0.000</td>
</tr>
<tr>
<td>Median</td>
<td>0.200</td>
<td>0.020</td>
<td>0.000</td>
</tr>
<tr>
<td>3rd Qu.</td>
<td>0.500</td>
<td>0.060</td>
<td>0.000</td>
</tr>
<tr>
<td>Max.</td>
<td>18.000</td>
<td>1.830</td>
<td>14.000</td>
</tr>
<tr>
<td>NAs</td>
<td>33.61%</td>
<td>0.14%</td>
<td>17.35%</td>
</tr>
</tbody>
</table>

Table 5.2: Workload data summary

5.2.2 Data preparation

Prior to packaging one needs to filter cluster tasks that apply to the considered aircraft specifications, cluster the tasks and fill in the missing values we discussed in the previous section. This can be done in any order. However, it is preferable to fill in the missing values prior to filtering because one can make a better guess for the missing values using more data. For example, if a prediction model is used for imputation. The final step is to 'pre-package' some clusters. Naturally, clusters with a total workload less than the user defined package capacity do not need to be split and are considered as a separate package. Only clusters for which the total workload exceeds this capacity need to be equalized.

Missing value imputation

We choose to impute the missing values in the preparation column by the column median because the preparation costs are concentrated around the median and contain only 3 outliers. The access column only has 4 missing values in total. Therefore, we simply impute the missing values with 0. The task costs have the most missing values and outliers. The missing task values were also imputed with the column median.

5.3 Results

5.3.1 Experimental setup

A specific aircraft model is picked at random from the MPD to serve as a case study so that we can demonstrate the performance of the proposed task packaging method. More specifically, 1273 tasks are filtered from the MPD belonging to Airbus A320 model 233 group 20-2C engine type IAE V2500 and APU type GTCP 36-300. The clustering method used prior to packaging is sort&cut with a maximum cluster width of 10%. The clusters are of various sizes providing us with a good
A variety of problem instances. A overview of each considered instance is provided in table 5.4. The ‘best’ and ‘worst’ columns indicate the total costs if all tasks are packed together or separated, respectively. Note that if we pack every task together the access costs are minimized. In case each task is packed separately the access costs are maximized. Thus, the difference between these two values is number of men hours that can be saved. Furthermore, the number of unique zones associated with the tasks including the number of tasks in each cluster can be found in columns $N$ and $L$, respectively. Finally, the packaging procedure is tested for three different package capacities, namely $S = (10, 20, 40)$.

<table>
<thead>
<tr>
<th>cluster</th>
<th>best</th>
<th>worst</th>
<th>% dev.</th>
<th>$N$</th>
<th>$L$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>109.30</td>
<td>112.67</td>
<td>2.99</td>
<td>118</td>
<td>157</td>
</tr>
<tr>
<td>2</td>
<td>110.21</td>
<td>112.47</td>
<td>2.01</td>
<td>67</td>
<td>120</td>
</tr>
<tr>
<td>3</td>
<td>51.00</td>
<td>57.42</td>
<td>11.18</td>
<td>67</td>
<td>54</td>
</tr>
<tr>
<td>4</td>
<td>97.15</td>
<td>100.93</td>
<td>3.75</td>
<td>56</td>
<td>87</td>
</tr>
<tr>
<td>5</td>
<td>97.10</td>
<td>98.12</td>
<td>1.04</td>
<td>42</td>
<td>28</td>
</tr>
<tr>
<td>6</td>
<td>152.50</td>
<td>152.50</td>
<td>0</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>7</td>
<td>41.24</td>
<td>42.97</td>
<td>4.03</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>8</td>
<td>90.06</td>
<td>91.98</td>
<td>2.09</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>9</td>
<td>91.22</td>
<td>93.10</td>
<td>2.02</td>
<td>73</td>
<td>68</td>
</tr>
<tr>
<td>10</td>
<td>65.23</td>
<td>67.82</td>
<td>3.82</td>
<td>77</td>
<td>54</td>
</tr>
<tr>
<td>11</td>
<td>178.42</td>
<td>180.33</td>
<td>1.06</td>
<td>57</td>
<td>73</td>
</tr>
</tbody>
</table>

Table 5.4: Cluster instances to be packaged

### 5.3.2 Performance

A few representative results are visualized in this section. Results obtained for cluster 4 are presented in figure 5.1. Observe that the package utilization is maximized because the number of packages used is minimized. Note that for $S = 20$ and $S = 40$ the tasks are packed optimally with regards to access costs while the packaging procedure was forced to split tasks with same access zones for $S = 10$. Because of this the obtained packaging solution for cluster 4 deviates 0.22 from the best case. Nevertheless, under the constraints the obtained solution is optimal.

Figure 5.2 shows the packaging results for cluster 11 which are similar to the results for cluster 4. Again, the package utilization is primarily maximized by minimizing the number of packages after which the optimal allocation of tasks into these packages is determined by the MILP solver. In addition, plot 5.2 illustrates how the packaging procedure deals with tasks that exceed the package capacity by themselves. Note that there are clusters only consisting of ‘outlier’ tasks, e.g. 8. Each tasks is considered a outlier for every considered $S$ and packed separately, see figure 5.3. Unless $S \geq 60$, the MILP solver will not be invoked.

Even if the packaging procedure always returns the minimum number of bins and the optimal allocation of tasks into these bins, a summary of the packaging results for all the considered clusters is provided in table 5.5.
Figure 5.1: Cluster 4 packaging results, dotted line indicates defined package capacity

5.3.3 Run-time

The largest test instance we presented is cluster 1 were \(N = 118, L = 157\) and \(M = 10\). The number of decision variables is equal to \(2750 \ (NM + LM)\) and the number of constraints is \(185, 427 \ (NML + M + L)\). Nevertheless, the MILP solver was able to find the optimal solution in 26 seconds on a 2.4GHZ single core processor machine.

Important to note is that the invoked MILP solver stops in less than a second if the approximated number of required packages is too little to obtain a feasible solution. Therefore our iterative search for the minimum number of required bins is time efficient. However, we did not compare our approach with the alternative approach where the objective function determines the minimal number of bins analogous to the bin-packing problem.
5.4 Conclusion

The optimization model for maintenance task packaging proposed in this chapter is able to obtain the optimal allocation of tasks with regards to the access costs and the optimal number of packages with regards to package utilization for the considered case. Generally, formulating the task packaging problem as an MILP proved to be an effective method to sub-set a given cluster of tasks efficiently into equalized maintenance task packages. However, the access time is a small part of the total maintenance time for each task. This means that even after optimally packing tasks the saved access time remains relatively small. Furthermore, the whole packaging process can be automated. This process includes preparing the data, constructing and solving the MILP.
Figure 5.3: Cluster 8 packaging results, tasks do not fit for considered values of $S$, MILP not invoked.

<table>
<thead>
<tr>
<th>cluster</th>
<th>opt. dev S=10</th>
<th>S=20</th>
<th>S=40</th>
<th>opt. $M$ S=10</th>
<th>S=20</th>
<th>S=40</th>
<th>$N$</th>
<th>$L$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>10</td>
<td>6</td>
<td>3</td>
<td>118</td>
<td>157</td>
</tr>
<tr>
<td>2</td>
<td>0.38</td>
<td>0.20</td>
<td>0.00</td>
<td>10</td>
<td>6</td>
<td>3</td>
<td>67</td>
<td>120</td>
</tr>
<tr>
<td>3</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>6</td>
<td>3</td>
<td>2</td>
<td>67</td>
<td>54</td>
</tr>
<tr>
<td>4</td>
<td>0.22</td>
<td>0.00</td>
<td>0.00</td>
<td>10</td>
<td>5</td>
<td>3</td>
<td>56</td>
<td>87</td>
</tr>
<tr>
<td>5</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>8</td>
<td>6</td>
<td>3</td>
<td>42</td>
<td>28</td>
</tr>
<tr>
<td>6</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>7</td>
<td>1.73</td>
<td>0.44</td>
<td>0.44</td>
<td>3</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>8</td>
<td>1.92</td>
<td>1.92</td>
<td>1.92</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>9</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>8</td>
<td>5</td>
<td>3</td>
<td>73</td>
<td>68</td>
</tr>
<tr>
<td>10</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>7</td>
<td>4</td>
<td>2</td>
<td>77</td>
<td>54</td>
</tr>
<tr>
<td>11</td>
<td>0.19</td>
<td>0.05</td>
<td>0.00</td>
<td>15</td>
<td>10</td>
<td>5</td>
<td>57</td>
<td>73</td>
</tr>
</tbody>
</table>

Table 5.5: Result summary
Chapter 6

Maintenance Scheduling

For the third and final phase of aircraft maintenance optimization, we investigate which methods are most suitable for maintenance scheduling. More precisely, we investigate which methods are capable of producing an efficient maintenance schedule for a fleet of aircraft given their flight plan and available maintenance opportunities over a planning horizon of at least two years. A maintenance schedule is considered efficient when the aircraft availability is maximized. Note that aircraft availability is maximized when the number of planned visits to or time spent at a maintenance hangar is minimized. Therefore, planning and performing preventive maintenance tasks between their 80 and 100% interval leads to an optimal schedule with regards to our objective. Two methods are presented in this chapter. The initial approach was to formulate the maintenance scheduling problem as an MILP and solve it to obtain the exact optimal schedule. Because of unforeseen problems with the exact method and the time constraint of this study a choice was made to implement and test a heuristic scheduling method instead. Nevertheless, the MILP formulation is presented and discussed in this chapter. Furthermore, the implemented scheduling method is used to test the effectiveness of packaging tasks prior to scheduling versus simply scheduling tasks individually with no regards to access synergy.

6.1 Methods

6.1.1 Exact solution

Similar to the task packaging problem from chapter 5, an MILP for task scheduling can be formulated using the bin-packing model as a basis. Two formulations are given below. The first one is limited to scheduling tasks for a single aircraft. The second one is an extension for fleet scheduling. The objective for both MILPs is to minimize the number of visits to a maintenance station, i.e. active maintenance opportunity. As a result this objective should also lead to an efficient workforce utilization.
Single Aircraft model

We define the notation for the maintenance scheduling of a single aircraft as following:

\[ M := \text{number of maintenance opportunities over time} \]
\[ L := \text{number of tasks/packages} \]
\[ S_k := \text{maximum workload for opportunity } k \]
\[ C_i := \text{total workload for task/package } i \]
\[ o_{ik} := \text{coincidence constant, based on the interval approximation} \]
\[ o_{ik} = 1 \text{ if the } 80-100\% \text{ interval of task/package } i \]
\[ \text{is reached when visiting opportunity } k \]

\[ x_{ik} := \text{binary decision variable} \]
\[ x_{ik} = 1 \text{ if task/package } i \text{ is scheduled at opportunity } k \]
\[ y_k := \text{binary decision variable} \]
\[ y_k = 1 \text{ opportunity } k \text{ is used} \]

We propose the following MILP formulation for the maintenance scheduling of a single aircraft:

\[
\begin{align*}
\text{min } & \sum_{k=1}^{M} y_k \\
\text{s.t. } & \sum_{i=1}^{L} C_i \cdot x_{ik} - S_k \cdot y_k \leq 0 \quad \forall k \\
& \sum_{k=1}^{M} x_{ik} = 1 \quad \forall i \\
& x_{ik} \leq o_{ik} \quad \forall i, k \\
& y_k \in \{0, 1\} \quad \forall k \\
& x_{ik} \in \{0, 1\} \quad \forall i, k
\end{align*}
\]

The auxiliary variable \( y_k \) is linked to the available capacity within restriction 6.2. Thus, the objective function 6.1 minimizes the number of used maintenance opportunities.

Important to note is that the number of tasks that need to be scheduled over a desired planning horizon have to be approximated using an assumed aircraft utilization over time. The reason is that certain maintenance tasks might need to be performed more than once during the planning horizon. For example, a monthly recurring task has to be performed at least 12 times during a time period of a year. This monthly recurring task can be seen as 12 tasks, each with a deadline in a different month. The restriction here is that each task recurrence needs to be scheduled within the 80-100% interval such that the resulting schedule is feasible. Recall the scheduling rule in chapter 2. Because of the recurrence of tasks, the set of tasks we need to schedule and assign to a decision variable \( x_{ik} \) is larger than the number of unique tasks. Furthermore, the calculated point in time at which the 80-100% interval overlaps with each maintenance opportunity has to be approximated for the model as well. Restrictions 6.2 and 6.3, respectively, make sure the workload at each opportunity in time is not exceeded and each task is scheduled once. Restriction 6.4 only allows tasks to be scheduled within their 80-100% interval. This last restriction is necessary to produce a solution that satisfies the scheduling rule for recurring tasks. In reality, this is a quite difficult constraint to satisfy because one needs enough capacity to be able to schedule every task and every recurrence of that task within the 80-100% interval.
Fleet extension

The notation for the extended model for scheduling of a fleet of aircraft is defined below. An important note is that we chose to label the tasks for each airplane in 1-dimension. We did not chose for a 2-dimensional formulation because the number of applicable tasks is different for each different airplane type within the fleet.

\[ M := \text{number of maintenance opportunities over time} \]
\[ N := \text{number of aircraft in fleet} \]
\[ L_r := \text{number of tasks/packages associated with AC } r \]
\[ L' := \text{total number of tasks, } \sum_{r=1}^{N} L_r \text{ where } L_0 = 1 \]
\[ A := \text{large constant, } A > \max_r L_r \]
\[ S_k := \text{maximum workload for opportunity } k \]
\[ C_i := \text{total workload for task/package } i \]
\[ o_{ik} := \text{coincidence constant, based on the interval approximation} \]
\[ o_{ik} = 1 \text{ if the 80-100\% interval of task/package } i \]
\[ K := \text{constant overhead cost} \]
\[ \text{arising from the time loss of setting up a airplane for maintenance} \]

\[ x_{ik} := \text{binary decision variable} \]
\[ x_{ik} = 1 \text{ if task/package } i \text{ is scheduled at opportunity } k \]
\[ y_k := \text{binary decision variable} \]
\[ y_k = 1 \text{ opportunity } k \text{ is used} \]
\[ z_{rk} := \text{auxiliary decision variable} \]
\[ z_{rk} = 1 \text{ if any task/package of aircraft } r \text{ is scheduled at opportunity } k \]

The proposed extended MILP formulation for scheduling of fleet maintenance tasks/packages is given below.

\[
\min \sum_{k=1}^{M} y_k \\
\text{add. s.t} \sum_{i=1}^{L'} C_i \cdot x_{ik} + \sum_{r=1}^{N} K \cdot z_{rk} - S_k \cdot y_k \leq 0 \quad \forall k \\
\sum_{k=1}^{M} x_{ik} = 1 \quad \forall i \\
x_{ik} \leq o_{ik} \quad \forall i, k \\
\sum_{i=L_{r-1}}^{L_r} x_{ik} \leq A \cdot z_{rk} \quad \forall r, k \\
\sum_{i=L_{r-1}}^{L_r} x_{ik} > A(z_{rk} - 1) \quad \forall r, k \\
y_k \in \{0, 1\} \quad \forall k \\
x_{ik} \in \{0, 1\} \quad \forall i, k \\
z_{rk} \in \{0, 1\} \quad \forall r, k
\]

For the fleet model the capacity constraint 6.2 is replaced by 6.5. The extended constraint incorporates the setup time before actual maintenance begins, i.e taxing the airplane to the maintenance
hub. The additional auxiliary variable $z_{rk}$ is used to count the number of unique aircraft with planned maintenance at the same maintenance opportunity. Restrictions 6.6 and 6.7 are added to express 6.8 as a set of linear equations.

$$z_{rk} = \begin{cases} 
1 & \text{if } \sum_{i=L_{r}-1}^{L_{r}} x_{ik} > 0 \\
0 & \text{otherwise}
\end{cases} \quad (6.8)$$

### Complexity

Finally, it is important to make some notes on the complexity of the proposed models. Table 6.1 shows that the complexity of this method increases by a factor of approximately $N$. This is in the optimistic case where we assume the number of maintenance opportunities remains the same.

<table>
<thead>
<tr>
<th>model</th>
<th># decision variables</th>
<th># constraints</th>
</tr>
</thead>
<tbody>
<tr>
<td>Single AC</td>
<td>$M + ML$</td>
<td>$M + L + ML$</td>
</tr>
<tr>
<td>Fleet</td>
<td>$M + MNL + NM$</td>
<td>$M + NL + MNL + 2NM$</td>
</tr>
</tbody>
</table>

Table 6.1: Model complexity
6.1.2 Heuristic solution

The developed scheduling heuristic is a algorithm that iteratively calculates, for each task, when the 80-100% interval is reached tries to schedule the task within its interval when the deadline is triggered. The algorithm requires an approximation of the aircraft utilization over time. Moreover, a flight schedule is can be used to approximate the utilization and the location of the aircraft at each point in time. This information is used to determine which maintenance hubs are feasible maintenance opportunities for different aircraft over time. Naturally, the workload capacities for all relevant maintenance hubs needs to be provided as well. The algorithm aims to assign all tasks/packages to as little maintenance opportunities as possible while making sure each tasks is planned before their deadline. The following steps are repeated, for each day, to construct an efficient and feasible maintenance schedule:

1. For each task/package, subtracting the remaining lifetime by the assumed (daily) utilization of each aircraft.
   - If remaining lifetime for the FH, FC or CH is equal to or less than 20% of the interval, label current location as feasible maintenance opportunity.

2. Determine which tasks/packages have exceeded their deadline (100% interval).
   - If remaining lifetime for the FH, FC or CH is equal to or less than zero, record that a deadline trigger occurs.

3. Schedule tasks/packages with a logged deadline trigger. the
   (a) Schedule the task/package at a previously labeled feasible opportunity with sufficient WL capacity and highest WL utilization so far, i.e at the opportunity where the WL sum of previously scheduled tasks is the highest.
   (b) If all labeled opportunities are empty schedule the task at the 100% interval.
   (c) If there is no more space at the labeled opportunities, search for a opportunity within 70-80% of its interval.
   (d) If there is still not enough space, pick the labeled opportunity with the lowest WL utilization and extend the capacity so that the relevant task can be scheduled.

4. All tasks with a deadline trigger are scheduled at the previous step. Thus, reset their FH, FC, CH accordingly.

An illustrative example how the algorithm works is given in table 6.2. The rows of each individual table are tasks or packages that need to be scheduled and the columns represent the point in time \( t \). Values are set to 1 when a specific point in time falls within the 80-100% interval. Furthermore, the last row of each sub-table indicates sum of scheduled task. In the example we assume a capacity of \( S = 3 \) at each opportunity and a WL of 1 for each task. Each row of sub-tables is a moment where at least one task deadline is triggered. The left-hand side sub-tables marks the occurrence of the deadline in bold-face while the right-hand side sub-tables indicate at what \( t \) the relevant tasks are scheduled with a circle. Note that this example is for a single aircraft, the actual implementation can schedule tasks for a entire fleet. Because different aircraft can be at different locations at every point in time, our implementation has a extra dimension such that the expected location of the maintenance opportunity can be logged. The location of each aircraft at each \( t \) is determined by the provided flight plan.

The proposed scheduling algorithm is greedy because it iteratively fills a single opportunity, according to first deadline first, with as many tasks as it can once that opportunity is used. The choice to schedule at 100% when scheduling at a previously unused opportunity is made to increase the chance that the interval of other tasks, that have not reached their deadline yet, overlaps with
this opportunity. This way the algorithm seeks to minimize the number of maintenance visits and as a result maximizes the workforce utilization at every maintenance opportunity.

### Improvement modules

The presented heuristic scheduling algorithm can be seen as a basis that can be extended by adding various heuristics or optimization modules to it such that the resulting schedule fits the goals of an operator. The modules can be simple human driven heuristics or more complex optimization schemes. A few examples of such improvement modules are given below.

1. Off-line (after algorithm terminates):

   (a) One can implement a function which improves the initial schedule by moving as many tasks from their 100% deadline to earlier opportunities without increasing the total number of used opportunities. The idea behind this function is that its quite risky to schedule something on the last day. Moving as many tasks/packages away from the 100% deadline makes the schedule more resistant to uncertainties that might occur in reality.

   (b) One can also implement a function that improves the greedy initial schedule by optimally swapping tasks for which the 80-100% interval overlap with each other.

2. On-Line (during scheduling procedure):

   (a) Larger tasks or packages suffer more from their loss in lifetime when planned outside their 80-100% interval than smaller tasks or packages. Because of this one can try to prioritize larger tasks during the scheduling procedure.

   (b) Alternatively one can try to prioritize tasks with the smallest 80-100% scheduling window.

During this study we tried out options 1a and 2a.

<table>
<thead>
<tr>
<th>t</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>S</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>t</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>S</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

### Table 6.2: Illustrative example of the heuristic scheduling algorithm
6.1.3 Comparison

A overview of the pros and cons of the heuristic and MILP approach to scheduling is provided in table 6.3. In practice one would need prior knowledge on how much resources are necessary to obtain a feasible solution using the MILP approach. Therefore, the exact method would not be able to obtain a feasible solution for many practical problems. Finally, note that the proposed heuristic approach overcomes the most important shortcomings (cons) of the MILP approach. Namely, that the MILP approach has a hard constraint for the interval planning and that it is not scalable to real life problems.

<table>
<thead>
<tr>
<th></th>
<th>Pros</th>
<th>Cons</th>
</tr>
</thead>
<tbody>
<tr>
<td>Exact solution</td>
<td>Can obtain optimal solution</td>
<td>Static, because of constraint that forces planning within 80 – 100% interval.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Slow and not scalable, requires much computational power</td>
</tr>
<tr>
<td>Heuristic solution</td>
<td>Dynamic, since it can deal with infeasible requirements</td>
<td>Greedy, however this can be corrected</td>
</tr>
<tr>
<td></td>
<td>Speed and scalability, low computational power requirements</td>
<td>No guarantee of optimality and no indication of difference with optimal solution</td>
</tr>
<tr>
<td></td>
<td>Extendable</td>
<td></td>
</tr>
</tbody>
</table>

Table 6.3: Pros and cons for the considered approaches

6.2 Task data

The data used for scheduling is the same data we previously used for packaging. A set of individual tasks and a set of tasks packed using methods discussed in the previous chapters are prepared and used to test the scheduling algorithm.

6.2.1 Data preparation

For this part of the study we split the task data into a set for short and long term scheduling. Tasks that recur more often than bi-weekly are considered short-term while tasks that recur less often than bi-weekly are considered to be long-term. The (simple) tasks that occur less than bi-weekly can be performed at line and require a more precise schedule then the other tasks that are performed at base. For this research we focus on the long term schedule which assumes the maintenance is done at base during the night. Thus, the resulting schedule indicates which day each task/package has to be performed. A different setup is necessary to schedule the short-term tasks because many short line maintenance opportunities exist during the day.

Furthermore, because we want to test the influence of packaging prior to scheduling in combination with our scheduling algorithm we prepare five data sets. The first two sets (1a and 1b) are prepared by clustering tasks using the cut&sort algorithm with parameter 10% after which we package the obtained clusters into packages with a maximum WL of 20 and 40. The other two data sets (2a and b) are obtained in the same way but with the cut&sort algorithm set to 0%. For the final set (3) we simply use the individual tasks without clustering or packaging.
6.2.2 Generated data

As mentioned previously, the robustness of the schedule can be improved by providing the algorithm with a expected aircraft usage over time. Because we have no such data available to us we generate a common seasonality trend and multiply it by our assumed average utilization for FH and FC. The generated seasonal trend and the resulting utilization over time are presented in figure 6.1a and 6.1b, respectively. Figure a shows a repeating seasonal trend for 4 years, while figure b is the result of multiplying the seasonal trend with the assumed utilization (FH & FC).

6.3 Results

6.3.1 Experimental Setup

To demonstrate that the proposed scheduling algorithm functions as intended and produces efficient schedules, a couple of test cases are set up. The first experiment involves a single aircraft that returns to the same location where base maintenance can be performed at the end of every day, see figure 6.2a. These experiments are performed on data sets with different clustering and packaging parameters to test the effects of packaging prior to scheduling. In addition the same experiment is repeated using a different prioritization policy during scheduling. Namely, larger tasks come first. Other relevant setup parameters can be found in table 6.4. The first year is simulated as a warmup for the algorithm after which a schedule for a planning horizon of 3 years is constructed. The capacity at the (only) maintenance hub is 40 WL. The assumed average daily usage of the aircraft expressed in FH and FC equals 8 and 4, respectively.

Finally, we demonstrate the performance of the algorithm on a small fleet of aircraft with three different base locations. The flight schedule is illustrated in figure 6.2b, each vertex indicates how the base location of each aircraft changes each day. More setup parameters for fleet scheduling are provided in table 6.4. The table is divided into 4 (row) sections and contains values for the single aircraft and fleet case separately (in columns). The first one contains settings regarding the planning horizon expressed in years. The second section lists the WL capacity $S$ of each of the
3 base locations. Each aircraft for the test case has a different age, this is expressed in the third section. Finally, the assumed average utilization (in FH & FC) is listed for each aircraft in the forth section. Furthermore, the task data set used for this experiment is the one of the sets used for the single AC experiments. More details can be found in table 6.5.

<table>
<thead>
<tr>
<th></th>
<th>Single AC</th>
<th>Fleet</th>
</tr>
</thead>
<tbody>
<tr>
<td>Warmup time</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Planning horizon</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>S</td>
<td>1</td>
<td>40</td>
</tr>
<tr>
<td>Base locations</td>
<td>2</td>
<td>40</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>20</td>
</tr>
<tr>
<td>age in years</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>8</td>
</tr>
<tr>
<td>FH</td>
<td>1</td>
<td>8</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>6</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>FC</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>2</td>
</tr>
</tbody>
</table>

Table 6.4: Test cases for scheduling algorithm

Figure 6.2: Considered flight networks for single AC and fleet scheduling

6.3.2 Single aircraft scheduling

Packaging influence

Scheduling results are presented in table 6.5 for 4 sets with different clustering and packaging parameters. The quality of the resulting schedule is measured in the total workload that is required to perform all the scheduled tasks and the number of used maintenance opportunities. We compare these results with instance 3, which has no clustering or packaging prior to scheduling, to determine if there are advantages to clustering and packaging. The most important finding can directly be observed from the resulting quality metrics, namely that the results for the tested
instances are very similar. While the total WL is slightly lower for instances 2a and 2b, the number of used opportunities (opps) is lower for instances 1a and 1b. Important to mention is that all the tasks/packages are scheduled within their 80-100% interval for all experiments.

<table>
<thead>
<tr>
<th>instance</th>
<th>rel. CW</th>
<th>max S</th>
<th>packages</th>
<th>total WL</th>
<th>opps.</th>
</tr>
</thead>
<tbody>
<tr>
<td>1a</td>
<td>10%</td>
<td>20</td>
<td>272</td>
<td>1118.8</td>
<td>49</td>
</tr>
<tr>
<td>1b</td>
<td>10%</td>
<td>40</td>
<td>251</td>
<td>1119.1</td>
<td>49</td>
</tr>
<tr>
<td>2a</td>
<td>0%</td>
<td>20</td>
<td>354</td>
<td>1112.8</td>
<td>51</td>
</tr>
<tr>
<td>2b</td>
<td>0%</td>
<td>40</td>
<td>333</td>
<td>1113.0</td>
<td>50</td>
</tr>
<tr>
<td>3</td>
<td>–</td>
<td>–</td>
<td>1273</td>
<td>1118.5</td>
<td>50</td>
</tr>
</tbody>
</table>

Table 6.5: Packaging influence

Prioritization policy

A comparison of the results for a first come first serve vs a large task first scheduling policy is given in table 6.6. A slight reduction in the number of opportunities used can be observed for instances where the data was clustered and packaged prior to scheduling. More precisely, the largest first policy lead to a reduction of the number of used opportunities (opps) by 1 and 2 for instances 1a and 2a, respectively. Thus, reducing the number of visits to a maintenance hub while still being able to perform all the tasks leads to a reduction in maintenance costs and increase of aircraft availability, which is the goal of this improvement module.

<table>
<thead>
<tr>
<th>instance</th>
<th>rel. CW</th>
<th>max S</th>
<th>packages</th>
<th>total WL</th>
<th>opps.</th>
<th>total WL</th>
<th>opps.</th>
</tr>
</thead>
<tbody>
<tr>
<td>1a</td>
<td>10%</td>
<td>20</td>
<td>272</td>
<td>1118.8</td>
<td>49</td>
<td>1118.7</td>
<td>48</td>
</tr>
<tr>
<td>1b</td>
<td>10%</td>
<td>40</td>
<td>251</td>
<td>1119.1</td>
<td>49</td>
<td>1119.0</td>
<td>49</td>
</tr>
<tr>
<td>2a</td>
<td>0%</td>
<td>20</td>
<td>354</td>
<td>1112.8</td>
<td>51</td>
<td>1113.2</td>
<td>49</td>
</tr>
<tr>
<td>2b</td>
<td>0%</td>
<td>40</td>
<td>333</td>
<td>1113.0</td>
<td>50</td>
<td>1113.1</td>
<td>50</td>
</tr>
<tr>
<td>3</td>
<td>–</td>
<td>–</td>
<td>1273</td>
<td>1118.5</td>
<td>50</td>
<td>1116.2</td>
<td>50</td>
</tr>
</tbody>
</table>

Table 6.6: Policy influence

6.3.3 Fleet scheduling

As mentioned previously, a case study was performed to test the functioning of the heuristic scheduling algorithm. The resulting schedule for the considered case is illustrated in figure 6.3. The workload cap is indicated with a dotted line while the scheduled WL is marked by a different shade of gray for each distinct aircraft. Red indicates the overhead costs when multiple aircraft are scheduled at the same day. One can observe that the algorithm functions as intended because it managed to schedule every task for all 4 aircraft without exceeding the desired WL at each location. With the exception of scheduling an outlier task, which has a WL of 120. The algorithm accordingly scheduled this task at an empty opportunity. What is not visible in the figure is that each task was scheduled within its 80-100% interval. Thus, the resulting schedule is optimal with respect to aircraft availability.

The flight operation of the considered case can also be observed from figure 6.3. Location 2 is where most airplanes meet and is thus the one with the highest average utilization when looking at the used maintenance opportunities. Location 1 is visited by one of two airplanes alternating by day and location 3 is visited by two planes every other day. This is also observable from the
resulting schedule. Finally, there is a single task that needs a WL of 120. This task was scheduled at an empty opportunity at the location 2.

Figure 6.3: Visualized schedule for the test case obtained using heuristic scheduling method.
6.3.4 Run-time

On a laptop with two 2.4Ghz cores, the heuristic scheduling algorithm takes less than 30 seconds to schedule around 1600 tasks/packages for a planning horizon of two years. The run-time scales linearly for larger problem instances because it simply needs more iterations when more aircraft, tasks, locations or a longer planning horizon are added.

6.4 Conclusion

The developed scheduling algorithm functions as intended and generates desirable schedules for the cases tested during this study. Furthermore, the results for the tested improvement modules show that there is room for improvement as suggested in subsection 6.1.2.

6.4.1 Future work

The methods identified and developed during this study form a the first steps towards a more efficient maintenance planning for commercial aircraft. Recommendation for further research are listed below with no particular order:

- Testing the effectiveness of task clustering and packaging prior to scheduling using various problem instances.
- Validate the developed methods on a real life problem
- Adapt the proposed heuristic scheduling algorithm to enable scheduling of line maintenance tasks
- Research possible improvements to the heuristic scheduling algorithm
- Development of a simulation model to aid in the development of robust schedules
- Development of a last minute schedule correction algorithm
- Test the proposed MILP approach for maintenance scheduling
Bibliography


