ReMAP

Real-time Condition-based Maintenance for Adaptive Aircraft Maintenance Planning

Deliverable D6.2

Deep reinforcement learning algorithm

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 769288

Document History

Revision Nr	Description	Author	Review	Date
v0.1	Initial draft	Pedro Andrade (UC), Bruno Santos (TUD) and Iordanis Tseremoglou (TUD)		27/04/2021
v0.2	Revised draft	Pedro Andrade (UC)		06/05/2021
v0.3	Addition of the algorithm comparison results	Pedro Andrade (UC)		26/05/2021
v0.4	Revised	Bernardete Ribeiro (UC) and Catarina Silva (UC)		30/05/2021
V1.0	Proof reading	Bruno Santos		31/05/2021

Index

1. Introduction

1.1. Project summary

ReMAP "Real-time Condition-based Maintenance for adaptive Aircraft Maintenance Planning" (hereinafter also referred as "ReMAP" or "the project"), is a European project started on the 1st of June 2018 and has a duration of four years. The project addresses the specific challenge to take a step forward into the adoption of Condition-Based Maintenance in the aviation sector. A data-driven approach will be implemented to achieve this, based on hybrid machine learning & physics-based algorithms for systems and data-driven probabilistic algorithms for systems and structures. A similar approach will be followed to develop a maintenance management optimisation solution capable of adapting to the real-time health conditions of the aircraft fleet. These algorithms will run on an open-source IT platform for adaptive fleet maintenance management. The proposed Condition-Based Maintenance solution will be evaluated according to a safety risk assessment, ensuring its reliable implementation and promoting an informed discussion on regulatory challenges and concrete actions towards the certification of Condition-Based Maintenance.

1.2. Purpose of this document

This document is the Deliverable D6.2 of the ReMAP project. It addresses the development of a deep reinforcement learning algorithm to produce optimised maintenance plans for a fleet.

The deliverable is part of the Work Package 6 (Maintenance Decision Support Tool) from the project. The work is related to Task 6.3 (Development of efficient machine learning).

1.3. Context

Aircraft maintenance is a relevant factor for the direct operating costs of an airline. Therefore, it is crucial to improve the current existing procedures and look towards maintenance optimisation. This document presents two approaches to produce efficient maintenance plans. The first is a comprehensive approach for scheduling routine maintenance tasks that contains a check scheduling phase and a task packaging phase. This algorithm differs from the traditional method of scheduling routine tasks in blocks. Instead, they are packaged individually into predefined maintenance slots. The second approach was developed to support the activities of Task 6.2 regarding stochastic maintenance scheduling. It deals with predictive-based tasks, and the goal is for the agent to learn the best moment to schedule these tasks under stochastic due dates. The algorithm also assumes that routine tasks are allocated in blocks.

1.4. Document structure

In Section 2, some background on reinforcement learning is presented, focusing on the Deep Q-learning algorithm used in the approaches reported in this document. Section 3 describes a comprehensive approach for scheduling routine maintenance tasks.

Section 4 presents a second approach for blocks and task allocation under a stochastic environment. A case study of different block interval policies is detailed in Section 5, and the testing of the scheduling algorithms with these new blocks is reported in Section 6. Finally, conclusions and lessons learned are discussed in Section 7.

2. Reinforcement learning

In Reinforcement Learning (RL), an agent interacts with the environment by performing a series of decisions to reach a particular goal, receiving a reward depending on the quality of the action chosen. The idea is that the agent will continuously learn by selecting actions that grant higher rewards. The training of the RL agent occurs in several episodes. An episode is a sequence of states that starts on an initial state and ends on a terminal state.

Formally, RL can be described as a Markov Decision Process (MDP). An MDP is a tuple (*S, A, P, R, γ*), where *S* represents the set of states, *A* is the action set, *P* is the state transition probability matrix, *R* is the reward function, and *γ* is the discount factor, which is used to give less relevance to rewards occurring far into the future. At each time step *t*, the agent interacts with the environment, choosing an action to take in the current state s_t. Then, it moves to the next state s_{t+1} and receives a reward r_t, which indicates the quality of the selected action (Figure 1)

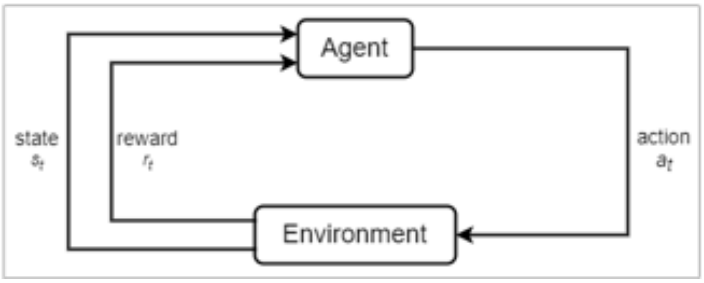


Figure 1: Agent-environment interaction (Sutton and Barto, 2018).

The goal is to find an optimal policy that corresponds to a sequence of actions that maximise the expected return, R_t. This return corresponds to the sum of discounted rewards over time, and it can be defined with the following equation, where r_t is the reward obtained at time *t* and $\gamma \in [0, 1]$.

$$R_t = \sum_{k=0}^{\infty} \gamma^k \, r_{t+k}$$

Another important element of RL is the value function, which defines the value of being in a given state. The value of a state *s* when following a policy π, denoted $v_\pi(s)$, is the expected return when starting at state *s* and following policy π thereafter, and it can be formally defined as:

$$v_\pi(s) = \mathbb{E}_\pi[R_t | S_t = s] = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k} | \, S_t = s\right]$$

There are two main methods for obtaining the optimal policy: policy iteration and value iteration. The first one aims to directly manipulate the policy, while the second aims to find an optimal value function and adopt a greedy policy. The Deep Q-learning algorithm used in both approaches is based on value iteration and is explained in the next section.

2.1. Deep Q-learning

Q-Learning (Watkins and Dayan, 1992) is one of the most popular RL algorithms. It makes use of a look-up table, called Q-table, which, in most cases, has the shape [states, actions]. Each number in the Q-table represents the quality of taking action, *a*, in a state, *s*, named the Q-value, *Q(s, a)*. At each time step, the RL agent observes the current state and chooses the action with a higher Q-value in that state. After acting, the agent receives a reward, which will be used to update the Q-table using the following equation, where *α* is the learning rate:

$$Q(s_t, a_t) = Q(s_t, a_t) + \alpha \left[r_{t+1} + \gamma \max_a Q(s_{t+1}, a_t) - Q(s_t, a_t) \right]$$

While a look-up table can be a simple and efficient solution for state-action spaces of reduced size, the same is not true for bigger and more complex problems. The traditional solution in these cases is the Deep Q-learning variant, which uses an Artificial Neural Network called Deep Q-network (DQN) to obtain approximations for the Q-values.

In RL, the agent learns from his own decisions, which means that in order to achieve a big reward and, at the same time, avoid converging to a local optimum, he must perform the largest number of decisions possible. This is known as the exploration-exploitation trade-off, where the exploration corresponds to the agent choosing a random action, and the exploitation corresponds to the agent selecting the action with the highest q-value in the current state. The ε-greedy strategy is used to ensure a proper exploration of the state-action space. The variable ε represents the probability of choosing a random action and is usually initialised to 1 with a decay rate over time. This ensures high exploration at the beginning and exploitation rising over time.

2.1.1. Deep Q-network

A Deep Q-Network (DQN) is a multi-layered neural network with weights θ. The input of the neural network is the state S_t and it has a fixed output layer with $|A|$ neurons, each representing the $Q(S_t, a_t, \theta)$ value from which it can be extracted the optimal policy. There are two main components in a DQN (Mnih et al., 2015). The first one is the target network with weights θ^- which is a delayed copy of the online network. The purpose of the target network is to update the loss function visible in the equation below.

$$L(\theta) = \mathbb{E}_{<s,a,r,s'>\sim \mathfrak{U}(R)} \left(r + \gamma \min_{a'} Q(s', a';\, \theta^-) - Q(s, a;\, \theta) \right)$$

where the first Q-term refers to the target network and the second Q-term to the online network. It is noteworthy that the weights of the target network remain fixed during τ steps to improve the learning stability, such that the online network can be calibrated in the direction of a stationary target. The second element of this algorithm is the experience replay buffer R, which is a memory where the agent stores the observed experiences. Then, at each timestep the agent randomly samples a batch of experiences to train the network. This helps to overcome two common problems of Deep Q-learning: the agent forgetting past experiences as time passes and the correlation between consecutive experiences. During the training phase, the weights θ of the online neural network are optimised with a stochastic gradient descent (SGD) algorithm to minimise the mean squared error of the loss function with respect to the weights θ of the online network (Hasselt et al., 2016).

The complete algorithm is outlined below in order to provide a clear sequence of the steps occurring in the training loop of the Deep Q-Learning agent. The weights of the neural network define the behavioural policy of the deep reinforcement learning agent. They

are updated with backpropagation via the SGD algorithm. The fundamental principle is to update the weights in the direction of the loss function gradient in order to decrease the overall error. The equation below formalises the neural network weights update.

$$\theta_{i+1} = \theta_i + \alpha \frac{\partial L(\theta_i)}{\partial \theta_i}$$

The backpropagation process is given in the following equation, with the loss function gradient of the Deep Q-Learning algorithm.

$$\theta_{i+1} \leftarrow \theta_i + \alpha(r + \gamma Q(s', a', \theta_i^-) - Q(s, a, \theta_i)) \cdot \nabla_{\theta_i} Q(s, a, \theta_i)$$

Algorithm 1 Deep Q-Learning

Initialize DQN with random weights θ
Initialize target network with DQN weights θ
for each *episode* **do**
  Reset Environment
  while not done **do**
    get state s
    calculate action-values $Q(s,a,\theta) \quad \forall a \in \mathcal{A}$
    select best action $a = \text{argmin}_a Q(s,a,\theta)$
    receive reward r
    transition to next state s'
    store experience $< s,a,r,s' >$ in $\mathcal{R}$
    for batch in BatchSize **do**
      sample experience $< s,a,r,s' > \sim \mathcal{U}(\mathcal{R})$
      if s' is terminal **then**
        target = r
      else
        target = $r + \gamma \min Q(s', a', \theta^-)$
      end if
      calculate loss $L_\theta = (Q(s,a,\theta) - \text{target})^2$
      update DQN $\theta_{i+1} \leftarrow \theta_i + \alpha \frac{\partial L(\theta_i)}{\partial \theta_i}$
    end for
  end while
  if τ is 10 **then**
    update target network $\theta_{i+1}^- \leftarrow \theta_{i+1}$
    $\tau_{i+1} \leftarrow 0$
  else
    $\tau_{i+1} \leftarrow \tau_i + 1$
  end if
end for

3. Comprehensive approach for routine maintenance tasks scheduling

This section presents the first developed approach for scheduling routine maintenance tasks. It does not follow the traditional method used in aviation maintenance of scheduling task blocks. Instead, the idea is to package tasks individually into predefined maintenance slots. The algorithm has two different phases: a check scheduling phase and a task packaging phase.

3.1. Check scheduling phase

Each aircraft must be grounded in the aviation industry to perform maintenance checks at certain intervals, as specified in the aircraft Maintenance Planning Document (MPD). These intervals are defined in flight hours (FH), flight cycles (FC), and calendar days (DY). Usually, each check type has its own maximum defined interval, and the corresponding usage metrics are reset to 0 immediately after a check is performed.

The goal of the check scheduling phase is to schedule A and C-checks for the fleet. An A-check is a lighter check that is performed approximately every two to three months and usually lasts one day. A C-check is a heavier check performed every 18 to 24 months and lasts more than one week. The best way to optimise this phase is to schedule checks as close as possible to their due date, that is, the date when the interval is reached. By doing this, we can achieve high aircraft utilisation and reduce the number of groundings in the long term.

We consider the following assumptions to define our long-term maintenance check scheduling approach:

A1: Aircraft utilisation is constant and can be obtained based on historical data.
A2: The duration of each check can be defined by the airline or estimated using historical data.
A3: There is a limited amount of hangar slots each day to perform A/C-checks.
A4: Each A/C-check uses only one hangar slot for its total duration.
A5: C-checks have a higher priority than A-checks.

The priority given to C-checks mentioned in **A5** is related to the fact that these are heavier checks that require the aircraft to be on the ground for several days. Thus, they have a relevant impact when scheduling A-checks because the aircraft usage parameters do not increase in that period.

The input data for the check scheduling phase contains the following information:

- Fleet initial conditions (total aircraft, elapsed time since previous checks)
- Fleet utilisation
- Check intervals
- Maintenance opportunities
- Maintenance elapsed times

The output consists of a list with the start and end times for the scheduled checks, along with some performance metrics, such as, the amount of interval unused.

As previously mentioned, a Deep Q-learning algorithm is used to optimise the scheduling of checks. Figure 2 shows an overview of the check scheduling phase. At each time step, the RL agent chooses an aircraft and schedules its next C-check on the closest available slot to the due date. The idea is to maximise the interval utilisation and reduce the number of groundings in the long term. If all C-checks are already scheduled for the entire horizon, the next A-check is scheduled instead. Then, there is the state transition, in which several variables are updated as a result of the latest scheduling. These include the due date for the next check, available hangar slots, and utilisation parameters. The algorithm ends when the next A or C-check due dates are beyond the horizon for all aircraft.

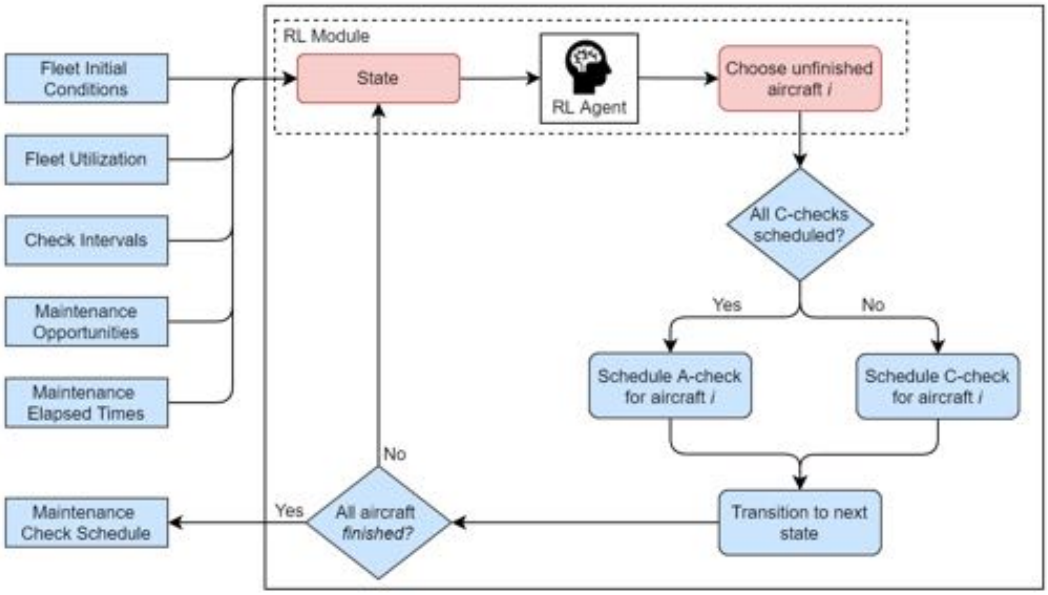


Figure 2: Overview of the check scheduling algorithm.

3.1.1. Reinforcement learning elements

When designing a reinforcement learning solution, defining its core elements such as the state, action set, and reward function is essential. The state must contain all relevant information for the agent to make decisions. In the check scheduling phase, the agent acts by choosing which aircraft will be scheduled next. At each step, it needs to access the updated schedule and state of the fleet to learn from the decisions made. Therefore, the state contains the following features, where $k \in \{A, C\}$:

- M_t^k: available maintenance slots for type k check on day t.
- D_i^k: due date of the next type k check of aircraft i.
- $FH_{i,t}^k, FC_{i,t}^k, DY_{i,t}^k$: cumulative utilisation in FH, FC, and DY, respectively, of aircraft i on day t for a type k check.

As mentioned before, at each step t, the agent selects an aircraft to have its next check scheduled. Thus, the action set can be described with $\mathrm{A_t} = i, \quad i \in \{1, \dots, N\}$, where i is the aircraft id, and N is the total number of aircraft.

The reward that the agent obtains depends on the amount of FH lost with the scheduling of a check. The reward function can be defined with:

$$R = \begin{cases} -\delta, & if\ check\ is\ scheduled \\ -10^5, & otherwise \end{cases},$$

where δ is the amount of FH lost with the scheduling of the last check. It includes two conditions that represent two penalty levels for the agent. The first condition corresponds to a smaller penalty equal to the FH lost when the check is scheduled successfully. The second condition is a higher penalty that punishes the agent if the check cannot be scheduled before its due date.

3.1.2. State transition

When transitioning from state S_t to state S_{t+1}, it is necessary to update some variables based on the last scheduling. The available maintenance slots during the period of the recently scheduled check are reduced by 1:

$$M_t^k = M_t^k - 1\ ,\ \ k \in K$$

After scheduling a type k check, the next due date corresponds to the day when the aircraft reaches the interval in one of the utilisation metrics. Thus, we can consider the following three variables that represent the limit date for each utilisation metric, where SD_i^k is the scheduling day resulting from the agent's action, and $(I_{i,k}^{DY},\ I_{i,k}^{FH},\ I_{i,k}^{FC})$ are the intervals in DY, FH, and FC for a type k check of aircraft i. The FH and FC limits are converted to a date by using the respective aircraft average utilisation fh_i and fc_i.

$$d_{i,k}^{DY} = SD_i^k + I_{i,k}^{DY}$$

$$d_{i,k}^{FH} = SD_i^k + \frac{I_{i,k}^{FH}}{fh_i}$$

$$d_{i,k}^{FC} = SD_i^k + \frac{I_{i,k}^{FC}}{fc_i}$$

The due date for the next type k check of aircraft i, D_i^k, is simply the minimum of these three limits:

$$D_i^k = \min\{\ d_{i,k}^{DY},\ d_{i,k}^{FH}, d_{i,k}^{FC}\ \}$$

Finally, the cumulative utilisation variables are set to 0 immediately after the scheduling day SD_i^k:

$$\begin{bmatrix} DY_{i,t}^k \\ FH_{i,t}^k \\ FC_{i,t}^k \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

3.1.3. Experimental setup

The check scheduling approach is tested with a fleet composed of 16 aircraft B787 for a planning horizon of 36 months. A-checks have an interval of 1500FH or 200FC. There are two slots every week to schedule A-checks, one on Tuesdays and another on Thursdays. Regarding C-checks, they have an interval of 18000FH or 36 months. There is a slot every workday to schedule C-checks, except in the summer season, between the 1st of June and 31st of August, when C-check work is interrupted. Historical data from the past five years is also used to calculate the average daily utilisation in FH and FC.

The agent is trained using experience replay, and the amount of exploration is controlled using an ε-greedy strategy. In the first episode, ε is equal to 1, meaning that the agent will always prioritise exploration. This value has a linear decay rate over each episode until it reaches 0.01. The Deep Q-network and hyperparameters for the Deep Q-learning algorithm were obtained using a grid search method. Several values were defined and individually tested for each parameter to see the ones that produced better results. The chosen network architecture consists of a multilayer perceptron with two fully connected hidden layers, the first with 200 neurons and the second with 100 neurons. Both layers have the sigmoid activation function and use the Adam optimiser (Kingma and Ba, 2014). The network weights are initialised with a normal distribution. There are two versions of the Deep Q-network: the online network and the target network. The first one is used to select actions greedily, while the second one is used to estimate the action values. The idea is to reduce overestimations that are known to occur when using the same network for action selection and evaluation. Table 1 presents the remaining hyperparameters.

Table 1: Deep Q-learning hyperparameters – check scheduling phase.

Parameter	Value
Learning rate	0.0001
Discount factor	0.95
Replay memory size	100,000
Initial exploration rate	1.0
Final exploration rate	0.01
Batch size	32
Target update frequency (steps)	1000
Training episodes	200

3.2. Task packaging phase

The goal of the task packaging phase is to package periodic tasks individually into maintenance slots. The check schedule obtained in the previous phase is used as input slots. Although, these checks might not be enough to incorporate all tasks. For that reason, smaller maintenance slots are created when necessary to ensure all tasks are packaged. Routine tasks must be performed at certain

intervals, which can be defined with the usual utilisation metrics: DY, FH, and FC. This interval can be used with the last execution date and the average aircraft utilisation to calculate the next due date of tasks.

We consider the following assumptions to define the task packaging problem:

A1: Aircraft utilisation is constant and can be obtained based on historical data.
A2: The available workforce for each slot is estimated from historical data.
A3: The workforce allocated to each slot has the necessary skills to perform the required maintenance tasks.
A4: Tools and parts are always available to perform maintenance tasks.

The input data for the task packaging phase contains the following information:

- Maintenance routine tasks (unique identifier, required access panels, interval, duration, skill)
- Access panels (unique identifier, opening time, closing time)
- Fleet utilisation
- Check schedule (from check scheduling phase)
- Estimated check workforce

The output is a maintenance plan containing the maintenance periods for each aircraft and the planned tasks in each one of them.

The task packaging algorithm works as follows: an open task is selected to be packaged at each step. A task can be considered as open if its next due date is within the defined horizon. Then, the candidate slots to package the task are retrieved. These correspond to the slots scheduled before the task due date with enough resources left to include the task. If there are no candidate slots to package the task, a new one is created. Otherwise, the RL agent chooses one of them. After a task is packaged, the due date of its next execution is computed, and the necessary maintenance variables are updated. An aircraft is finished when all its tasks are scheduled for the entire horizon. In other words, when all of its tasks have a due date beyond the horizon. This process is repeated for every aircraft of the fleet. Figure 3 shows an overview of the task packaging phase.

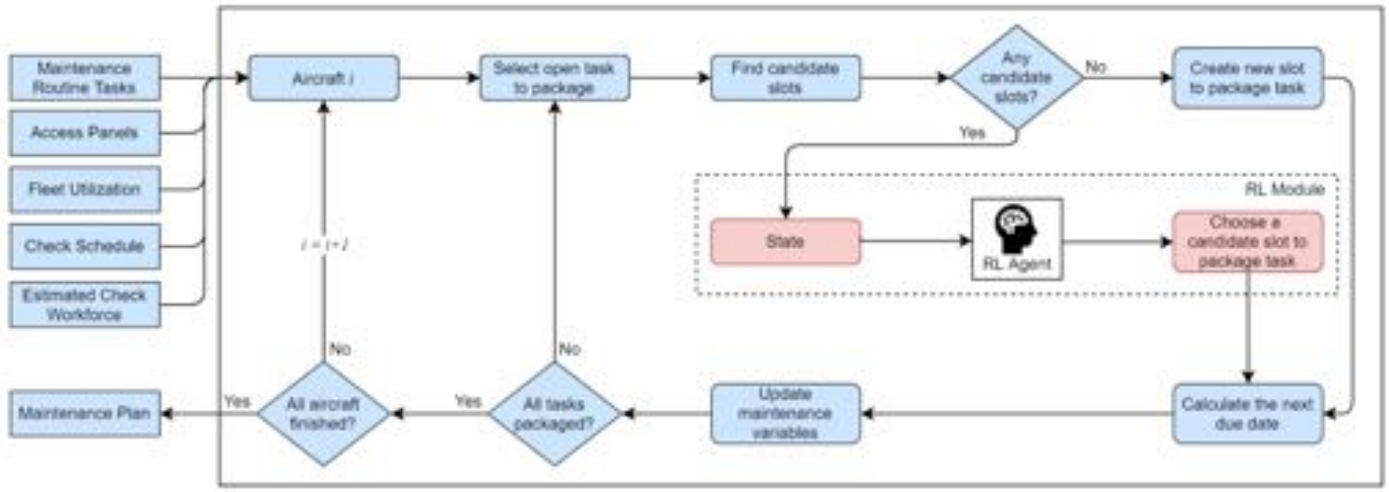


Figure 3: Overview of the task packaging algorithm.

Several conditions must be followed when creating one of the smaller maintenance slots. The chosen period for the slot corresponds to the closest period before the task due date that meets the following conditions:

- It must be scheduled on a workday.
- There is a maximum number of grounded aircraft at the same time to ensure operation demands are always satisfied. A new slot cannot be scheduled for periods that have reached this limit.
- There is a minimum number of days between two consecutive groundings for the same aircraft. The idea is to balance the fleet maintenance requirements and avoid cases where the same aircraft is grounded multiple times in the same week.

It is also important to note that tasks are only packaged into smaller slots if no checks are available. The aim is to reduce the number of groundings by doing most of the work at A and C-checks.

The key part of this algorithm is the selection of the slot in which a task is packaged. When there is more than one candidate slot, the RL agent is responsible for that selection. The two metrics used in the agent's decision are the task interval utilisation and the task access cost. Ideally, tasks are packaged into slots that are close to their due date. By choosing those slots, the task interval utilisation is maximised, and fewer tasks are performed in the long term. On the other hand, it is also important to reduce access costs by packaging together tasks with the same panel requirements.

3.2.1. Reinforcement learning elements

The role of the agent in this phase is to choose the slot to assign to each task. It needs to have enough information regarding the task being packaged and the candidate slots. Thus, the state contains the following variables, for all c representing the candidate slots:

- $D_{i,j}$: due date of task j for aircraft i.
- $AP_{i,j}$: access panels required for task j of aircraft i.
- SD_i^c: start date for candidate slot c of aircraft i.
- W_i^c: workforce remaining in candidate slot c of aircraft i.
- AP_i^c: access panels required for all tasks packaged on candidate slot c of aircraft i.

As mentioned before, the agent makes decisions based on the task utilisation and access cost, which translates directly into two actions. One of them is to package the task in the candidate slot closer to its due date, increasing the interval utilisation. The other action is to choose the slot containing the higher number of tasks already packaged with the same panel requirements.

The reward function determines the reward that the agent receives after making a decision. In this problem, it is defined according to the goals of maximising interval utilisation and minimising the access cost:

$$R = \begin{cases} \dfrac{shared\ access\ hours}{total\ access\ hours} - \%\text{interval unused} \times \text{task duration}, & if\ task\ requires\ access \\ -\%\text{interval unused} \times \text{task duration}, & otherwise \end{cases}$$

The first case of the reward function is used if the task being packaged requires at least one access panel to be opened. It contains a positive term equal to the percentage of shared access hours with other tasks in the selected slot. These hours correspond to the opening and closing times of the respective panels. The goal of maximising interval utilisation is represented in the second term,

which gives a negative signal equal to the percentage of interval unused multiplied by the task duration. This duration is included because higher duration tasks usually have higher costs. These tasks should have a higher contribution for the reward because a lower interval utilisation means they need to be performed more times in the long term. The second case is for tasks without access requirements as it only contains the term regarding the interval utilisation goal.

3.2.2. State transition

After packaging a task in one of the candidate slots, several variables need to be updated. The next due date of the task is calculated in a similar way to the check due dates in the previous phase. The limits in DY, FH, and FC are obtained with:

$$d_{i,j}^{DY} = SD_i^c + I_{i,j}^{DY} \ ,$$

$$d_{i,j}^{FH} = SD_i^c + \frac{I_{i,j}^{FH}}{fh_i} \ ,$$

$$d_{i,j}^{FC} = SD_i^c + \frac{I_{i,j}^{FC}}{fc_i} \ ,$$

for a task j of aircraft i packaged on a slot c, where $[I_{i,j}^{DY},\ I_{i,j}^{FH},\ I_{i,j}^{FC}]$ are the task intervals in the three metrics and $[fh_i, fc_i]$ is the average daily aircraft utilisation. The due date of task j corresponds to the date when the first limit is reached:

$$D_{i,j} = \min\{\ d_{i,j}^{DY}, d_{i,j}^{FH}, d_{i,j}^{FC}\ \}$$

The access panels required for the execution of the task are added to the panels being opened in the selected slot c, AP_i^c. The workforce remaining on that slot is also updated. However, there are two cases for this update: (1) the task requires at least one access panel; (2) the task requires no access panels.

$$W_i^c = \begin{cases} W_i^c - (\ l_j + \ OT_p + \ CT_p) \ , & (1) \\ W_i^c - l_j \ , & (2) \end{cases}$$

Variable l_j is the duration of task j, OT_p is the opening time of panel p, and CT_p is the closing time of panel p, for all p representing the panels that are not shared between the task and the selected slot.

3.2.3. Experimental setup

The task packaging phase is tested with the same fleet of 16 aircraft B787 for a planning horizon of 12 months. A total of 729 routine maintenance tasks are packaged, of which 186 are part of A-blocks defined by the airline, and the remaining 543 are part of the C-blocks. The check schedule obtained in the previous phase of this approach is used as an input in this phase to define the set of maintenance slots in which to package the tasks. When attempting to schedule a smaller slot to package a task, a condition is set to guarantee no other grounding for the aircraft within four workdays. Additionally, to ensure operation demands, the maximum number of aircraft groundings in parallel is set to 2. Historical data from the past five years is used to calculate the average daily utilisation of the fleet and the estimated amount of workforce required for each type of slot.

The deep Q-network architecture is identical to the check scheduling phase. It consists of a multilayer perceptron with two fully connected hidden layers, the first with 200 neurons and the second with 100 neurons. Both layers have the sigmoid activation function and use the Adam optimiser. The hyperparameters were obtained with a grid search method and are detailed in Table 2.

Table 2: Deep Q-learning hyperparameters – task packaging phase.

Parameter	Value
Learning rate	0.0001
Discount factor	0.5
Replay memory size	100,000
Initial exploration rate	1.0
Final exploration rate	0.01
Batch size	32
Target update frequency (steps)	1000
Training episodes (T)	200

4. State space discretisation for blocks and task allocation

The second approach was developed considering the division of the task intervals into segments that describe the state of each task or blocks of tasks. The approach also relies on a deep reinforcement learning algorithm. It was developed to address the problem of considering the assessment of Condition-based Maintenance (CBM) policies in the airline maintenance scheduling problem (AMSP). It was considered a transitional period from preventive to predictive maintenance by considering the same problem blocks of routine tasks and individual tasks driven by prognostics, which we will call CBM Tasks. The first ones are clustered during the task-packaging phase in maintenance checks, and they must be performed periodically based on a usage interval.

On the other hand, CBM tasks are monitored individually during the scheduling process using prognostics. Thus, we define a transition strategy that combines traditional letter checks, predefined in the task-packaging problem, with a task-based methodology for CBM tasks. The scheduling model allocates A-checks to maintenance opportunities and inserts CBM tasks into the scheduled A-checks based on the RUL prognostics.

The problem definition is provided in the first subsection, followed by the methodology developed to solve the problem. The final subsection presents results that validate the efficiency of this Approach 2.

4.1. Problem Definition

The mathematical formulation of the airline maintenance scheduling problem (AMSP) depends on the choice of the optimisation model, the planning horizon, and the uncertainty of the factors involved. The proposed formulation is inspired by the works of Deng et al. (2020) and Lagos et al. (2020) that employ a dynamic programming approach and a look-ahead estimation of the next states with the objective of maximising aircraft utilisation. We also leverage the approximate dynamic programming (ADP) framework presented in the work of Powell and Topaloglu (2006). Their formulation recognises two key elements in any dynamic resource allocation problem: the resources and the demand. In the AMSP, resources are identified as the maintenance opportunities predefined in the slot availability calendar, while the demand corresponds to the scheduling units, also called routine blocks and CBM tasks, that need to be allocated. Both types of scheduling units are modelled in the exact same way. However, routine blocks have a due date dictated by an interval, while the due dates of CBM tasks are dependent on RUL prognostics. The formulation makes use of the following five main assumptions.

1. **Aircraft utilisation is known and constant**. Aircraft routings are usually only known three to five days before operations. Therefore, a statistical analysis is performed to calculate the average flight hours and cycles per day to produce monthly A-check schedules.
2. **The amount of weekly slots is constant, and there are sufficient labour hours**. We assume that the size of the A-checks will be similar, and the CBM tasks can be accommodated within the scheduled maintenance slot.
3. **Inventory management is not considered**. Tools and parts are assumed to be always available to perform scheduled maintenance tasks.
4. **CBM tasks can only be allocated within a routine block**. In practice, an individual task could also be performed in line maintenance or additional slots besides A-checks.
5. **Additional tasks can be performed with a buffer time**. Eventual non-routine tasks and unscheduled maintenance events can be accommodated within the A-checks.

The goal is to select a maintenance slot date for every aircraft in order to maximise fleet utilisation. The problem is set up in a sequential fashion such that at every timestep t, the scheduling unit with the highest priority (ACt) is selected based on the earliest due date. The problem is deemed solved when the model schedules the last routine block or CBM task on the time horizon date (TH). Every time an aircraft is selected, either a routine block (A-check) or a CBM task can drive the earliest due date selection. In the first case, a routine block needs to be allocated to a maintenance slot, while in the second case, a CBM task is inserted in one of the A-checks previously scheduled. The problem formulation uses the notation presented below. The choice of reinforcement learning (RL) or approximate dynamic programming (ADP) requires formulating the problem as a sequential decision-making process, most commonly as a Markov Decision Process (MDP). An MDP assumes the presence of the state-space, the action-space, the reward function, and the transition function. In the remainder of this section, a clear overview of each element is provided.

4.1.1. State Space

The maintenance planning simulation occurs in a sequential process by selecting the most critical scheduling unit, either a block or a CBM task, depending on which has the earliest upcoming due date. Each item or *scheduling unit* has a time window that goes from the previous execution to the next due date, dictated by either an interval or a prognostic. This time window is discretised in $|A|$ sub-intervals, as shown in Figure 4, to analyse the available resources in each of them, as well as the demand for those resources. The state vector refers to the time window of the most critical item to calculate the relevant environment features. A total of six features is employed to capture the scheduling environment condition through which the model can learn the effect of its decisions:

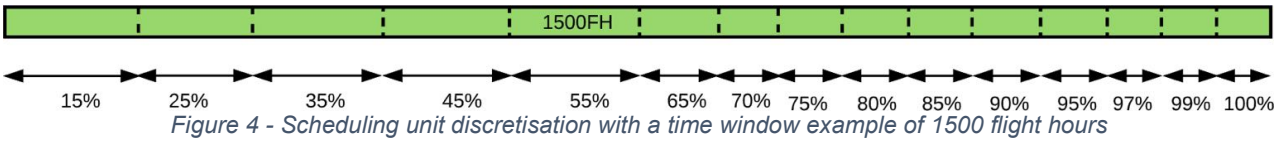


Figure 4 - Scheduling unit discretisation with a time window example of 1500 flight hours

- $D_{t,a}$: demand or number of scheduling units competing for a slot in the sub-interval a. If there is no slot available, the value of this feature becomes null.
- $R_{t,a}$: resources or number of the available slots to the demand Dt,a competing for the slot in sub-interval a of the most critical unit AC_t at time t.
- $lh_{t,a}$ is the average estimated reward for the competing aircraft computed with a look-ahead function, assuming that the slot in sub-interval a is occupied by AC_t. This feature aims to capture the performance consequences for the rest of the fleet when a slot is being considered for the most critical scheduling unit. The function works with a greedy policy: every time a slot in sub-interval a is considered, it is removed from the available resources. The function assigns the latest possible slot to each aircraft in their priority order and calculates the cost of allocating the remaining resources to the competing aircraft.
- $h_{t,a}$indicates the time index of the slot in the simulation horizon, such that the system is aware of the slot position in time. This feature represents the relative position of the maintenance date with respect to the simulation end date.
- $p_{t,a}$ is the probability of failure at each sub-interval a based on the RUL prognostics of a CBM task. In the case of routine blocks, the problem is deterministic, and this feature is nullified.
- $b_{t,a}$ is a binary variable that indicates whether the scheduling unit AC_t is a routine block or a CBM task. This feature takes the same value for all $a \in A$.

At every iteration, the features are calculated for each discretised sub-interval $a \in A$, and the resulting vector becomes a column of the state. The final state given has a size of $|A| \times 6$ and is given by $S_t = (R_t, D_t, lh_t, h_t, p_t, b_t)$ for all sub-interval.

4.1.2 Action Space

The action $a_t \in$ A controls the scheduling environment by determining the aircraft utilisation rate before entering maintenance. Thus, the action space is given by the discretised sub-intervals of the scheduling unit, plus the additional possibility of having an aircraft on the ground (AOG). In the latter case, the aircraft is forced to be grounded and wait for a maintenance opportunity. Therefore, the complete action-space with the discretisation scheme shown in Figure 5 becomes:

$$A = \{15\%, 25\%, 35\%, \dots, 95\%, 97\%, 99\%, 100\%, AOG\}$$

Let Figure 5 be a visualisation of the maintenance scheduling environment, with the available slots on top and the scheduling unit intervals of different aircraft on the bottom. The interval of AC1 on top of the list is the most critical as it has the earliest due date. Moreover, a total of five slots, indicated with a red "X", are available during the time window of this scheduling unit. However, only four actions are feasible based on the discretisation scheme of the interval. For clarity purposes, the four feasible actions have been highlighted with a striped pattern. The second feasible action contains two slots, and if it were selected, the latest slot would be assigned to the most critical scheduling unit.

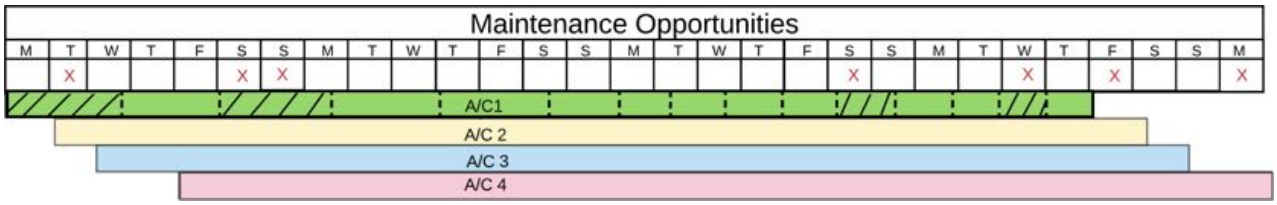


Figure 5 - Maintenance scheduling environment representation

There are two types of scenarios that can occur. When a routine block has the earliest due date, maintenance opportunities are given based on hangar availability. On the other hand, when a CBM task needs to be allocated, the opportunities are determined by the slots occupied with a routine block of the same aircraft tail. In this way, CBM tasks are inserted in the same slot as the A-checks. Ideally, the maintenance scheduling units should be allocated as close as possible to their due date in order to maximise aircraft utilisation. Nevertheless, action is feasible only if a maintenance opportunity overlaps with the respective sub-interval, as shown in Figure 5 with the striped pattern. Thus, the feasible region A_t at time t is expressed as follows:

$$A_t = \begin{Bmatrix} a_i : PE_t^{AC_t} + a_{i-1} \cdot I_t^{AC_t} \leq MD_i^{AC_i} \leq PE_t^{AC_t} + a_i \cdot I_t^{AC_t} \ , & \forall\, i \in A \\ a_i \leq M_i^{AC_i} \ , & \forall\, i \in A \end{Bmatrix}$$

The first constraints ensure the maintenance slot date is in the desired sub-interval $[a_{i-1}, a_i]$, which is calculated based on the previous execution date ($PE_t^{AC_t}$) and the interval requirement ($I_t^{AC_t}$) of the maintenance scheduling unit. Likewise, the second set of constraints consider action a_i to be feasible only if at least one maintenance opportunity is present at the time the aircraft is utilised by a_i percentage. As an example, the equation shows the feasible action space of the example presented in Figure 5.

$$A = \{15\%, 35\%, 90\%, 99\%, AOG\}$$

4.1.3 Reward Function

In reinforcement learning, the sum of the rewards received over an entire episode determines the final objective function value, which is referred to as the cumulative reward. The evaluation of the action a_t, given the state S_t and the next state S_{t+1}, is assessed through the rewards received at each timestep. In this way, the model can extract information from the intermediate steps between the start and the end of an episode. We interpret the reward as a cost function that needs to be minimised by the model applied to the problem such that the fleet utilisation is maximised.

The operational impact of the actions represents the interval limit consumed by an aircraft before entering the maintenance hangar. The linear sum of the interval utilisation values would not lead to maximising the whole fleet utilisation because some aircraft might have extremely high rates, while others might be utilised much less. Since the objective function would not reflect this behaviour, an alternative approach is chosen to maximise fleet utilisation through the cumulative reward function. A logarithmic transformation of the interval utilisation can still represent the cost of individual actions but, more importantly, the minimisation of the cumulative reward would lead to the maximisation of the overall fleet utilisation. The set of equations below shows the cost or reward function shape, where the constant K takes value 1 for CBM tasks, while the K value of routine blocks is doubled in order to outline the hierarchy of a task-cluster. The cost function is shaped such that it is minimised when all the actions are as close as possible to 95%, as the airline strives to have a 5% margin due to the variability in aircraft utilisation. Lastly, the action that is penalised most heavily corresponds to an *aircraft on the ground* (AOG) or missed maintenance opportunity. This option has been included for feasibility purposes, such that if all maintenance opportunities are occupied or the due date of a task occurs before the prescribed maintenance date, a large penalty is returned in order to respect the scheduling unit's interval.

$$R_t(S_t, a_t) = \begin{cases} Kln\left(\frac{100}{a_t}\right) & if \quad a_t \leq 95\% \\ Kln\left(\frac{100}{2.95 - a_t}\right) & if \quad 95\% \leq a_t \leq 100\% \\ 2Kln(100) & if \quad a_t \; is \; AOG \end{cases}$$

4.1.4. Transition Dynamics

The transition function dictates the dynamics of the environment from state S_t to state S_{t+1}. It is noteworthy that, at each timestep, the agent processes the aircraft routine block or CBM task with the highest priority and decides to which opportunity it should be allocated. The complete set of steps involved in the simulation process are outlined as follows:

- Simulate the fleet utilisation
- Update remaining useful life (RUL) for all aircraft
- Calculate the next due date of all scheduling units
- Select the most urgent scheduling unit based on the earliest due date
- Process features for the selected scheduling unit
- Select a scheduling action
- Update environment attributes
- Repeat the previous steps until the end of the horizon

Every time a scheduling unit AC_t is being processed in state S_t, a scheduling action a_t* is chosen based on the policy of the reinforcement learning model. Subsequently, the maintenance resources are updated by subtracting the slot that has just been allocated,

$$M_{a_t*}^k = M_{a_t*}^k - 1 \quad , \quad \forall \ k \in K$$

and the scheduled date variable ($SD_t^{AC_t}$)becomes the date of the chosen slot.

$$SD_t^{AC_t} = MD_{a_t*}^{AC_t}$$

Once the current aircraft has been scheduled, the whole fleet utilisation is simulated for a number of days that goes from the previous reference date RD_t^k until the selected scheduled date $SD_t^{AC_t}$. The simulated days are converted to flight hours and flight cycles according to daily ratios (δ_{fh}^t, δ_{fc}^t) that are estimated based on the calendar month at timestep t for each scheduling unit k.

$$\begin{bmatrix} \Delta DY_t^k \\ \Delta FH_t^k \\ \Delta FC_t^k \end{bmatrix} = \begin{bmatrix} SD_t^{AC_t} - RD_t^k \\ \Delta DY_t^k \cdot \delta_{fh}^t \\ \Delta DY_t^k \cdot \delta_{fc}^t \end{bmatrix} \quad \forall \ k \in K$$

These parameters are used to update the remaining calendar days (CY), flight hours (FH), and flight cycles (FC) of the whole fleet.

$$\begin{bmatrix} DY_{t+1}^k \\ FH_{t+1}^k \\ FC_{t+1}^k \end{bmatrix} = \begin{bmatrix} DY_t^k \\ FH_t^k \\ FC_t^k \end{bmatrix} - \begin{bmatrix} \Delta DY_t^k \\ \Delta FH_t^k \\ \Delta FC_t^k \end{bmatrix} \quad \forall \ k \in K$$

However, the remaining utilisation parameters of the scheduling unit ACt, that has just been allocated are re-initiated to their original interval limitations.

$$\begin{bmatrix} DY_{t+1}^{AC_t} \\ FH_{t+1}^{AC_t} \\ FC_{t+1}^{AC_t} \end{bmatrix} = \begin{bmatrix} I_{DY}^{AC_t} \\ I_{FH}^{AC_t} \\ I_{FC}^{AC_t} \end{bmatrix}$$

A limitation of this approach is the assumption of complete aircraft utilisation information. Therefore, it does not include future uncertainty related to the utilisation variability, for this reason, a 95% target utilisation was defined. The remaining useful life variables (X_t^k) are updated with the minimum driving requirement either in DY, FH or FC.

$$X_t^k = min\left(\begin{bmatrix} DY_t^k \\ FH_t^k / \delta_{fh}^{t,k} \\ FC_t^k / \delta_{fh}^{t,k} \end{bmatrix} \right) \quad \forall \ k \in K$$

Then, the routine block due dates are updated based on the remaining interval days, while in the case of a CBM task, a prognostic ω_k dictates the scheduling unit due date.

$$DD_{t+1}^{k} = \begin{cases} RD_t^k + X_t^k & if\ k\ is\ a\ routine\ block \\ \omega_t^k & if\ k\ is\ a\ CBM\ task \end{cases} \qquad \forall\ k\ \in K$$

The transition dynamics are slightly different when a CBM task needs to be allocated. Since it is necessary to simulate the prognostic multiple times with varying uncertainty, the CBM task scheduling occurs in multiple steps. At every iteration, the model updates the reference date to the following day (RD_{t+1}^{k}) to simulate a chronological sequence of actions until the next available maintenance opportunity. On the other hand, the reference date for the routine blocks is directly updated to the selected scheduled date.

$$RD_{t+1}^{k} = \begin{cases} SD_t^{AC_t} & if\ k\ is\ a\ routine\ block \\ RD_t^k + 1 & if\ k\ is\ a\ CBM\ task \end{cases} \qquad \forall\ k\ \in K$$

In this way, the prognosticated due date (ω_k) is re-sampled with less uncertainty since the model gets closer to the end of the interval. The CBM task is considered to be allocated only when the reference date (RD_{t+1}^{k}) is the same as the scheduled date (SDACt). The reward of the intermediary steps in the CBM task allocation process is set to zero until the scheduling date is reached. Furthermore, the previous maintenance execution of ACt and the minimum interval for every unit in K, are updated in the equations below.

$$PE_{t+1}^{AC_t} = SD_t^{AC_t}$$

$$I_{t+1}^{k} = DD_{t+1}^{k} - PE_{t+1}^{k} \qquad \forall\ k\ \in K$$

Lastly, the scheduling units are positioned in descending order, based on the updated due dates, and the next most urgent one is selected.

$$AC_{t+1} = \underset{k}{\operatorname{argmin}} \left(\left[DD_{t+1}^{k} \right]_{k \in K} \right)$$

Figure 6 shows the transition dynamics that allow the agent to move from slot to slot. At every timestep, a new scheduling unit AC_t is selected based on its due date. The action chosen determines the opportunity to which the respective scheduling unit is allocated, as well as the simulated days of fleet utilisation. In the case of S_{t+3}, the aircraft selected cannot be scheduled after the reference date. For this reason, the sequence of actions takes a step backwards in the schedule. It is noteworthy that the timestep index indicates the chronological order in which the actions are taken. The actual date and time are determined by the date of the maintenance slot that is selected with every action.

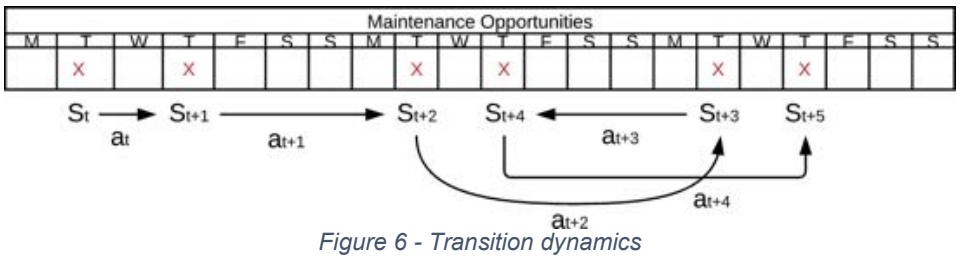


Figure 6 - Transition dynamics

4.2. Methodology

To optimise the maintenance task allocation and increase aircraft utilisation, the proposed approach allocates maintenance opportunities to both routine blocks and CBM tasks with a reinforcement learning (RL) agent. The scheduling process is sequential. First, a routine block or A-check must be allocated to an available slot. Then, the CBM tasks corresponding to the same aircraft tail can be inserted in maintenance opportunities defined by A-checks.

The goal of reinforcement learning is to develop a policy for sequential decision problems by optimising the cumulative discounted reward signal in the following equation

$$G_t = R_{t+1} + \gamma R_{t+1} + \gamma^2 R_{t+1} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$$

where G_t is the cumulative discounted reward, R_t is the reward per time segment and γ is the discount factor. The discount factor $\gamma \in [0, 1]$ assigns the importance of future rewards with an exponential factor with respect to the current timestep, such that the future returns are of less value than current rewards. The choice of discount factor requires careful consideration about the relevance of future rewards for the current state. It is important to consider the effect of a scheduling action on the short-term because it may compromise opportunities for the rest of the fleet and the long-term. After all, a scheduling action influences the start of the following interval.

4.2.1. Deep Q-Learning network

The state-space of the AMSP is extremely large, and the solution space is non-convex. The dimensionality problem can be circumvented with a neural network as a non-linear function approximator that learns a dynamic scheduling policy. Moreover, it is not required to sweep across all the state-action pairs to achieve convergence. In this way, the neural network facilitates forward dynamic programming (Powell, 2011): it replaces the computational burden of looping through each possible state-action pair with the statistical problem of estimating their value.

The DQN architecture is designed in function of the state-space and the action-space of the AMSP. The state-space defines the size of the input layer, while the action-space determines the size of the output layer. A visualisation of the DQN structure is shown in Figure 7, alongside the shape of each layer and the parameters that have to be optimised in the model. The model is composed of a total of five layers. The first one is labelled the input layer and has the same size as the state vector and, for this reason there are no model parameters because it simply receives an input from the RL environment. The following two layers, also called dense layers, have 100 neurons each, and their output shape depends on the previous layer. The first dense layer has 700 parameters corresponding to 6 features times 100 weights plus 100 biases of each neuron (6.100+100 = 700). Each neuron provides an abstract feature; therefore, there are 100.100 + 100 = 10 100 parameters in the second layer. Moreover, the flatten layer simply makes a one-dimensional vector out of the previous layer. Lastly, the output layer has a size of 101 neurons, the same as the action space. The total parameters of this layer are 10 000.101 + 101 = 1 010 101.

The neural network weights are initialised with a normal distribution to reduce bias. Moreover, the activation function for each layer is called swish (Ramachandran et al., 2017). Instead, the last two layers are designed with a linear activation function because the

DQN output represents the Q-value of the next state, and it should converge to the discounted reward value. The learning rate, the exploration decay and the discount factor were selected upon a careful sensitivity analysis.

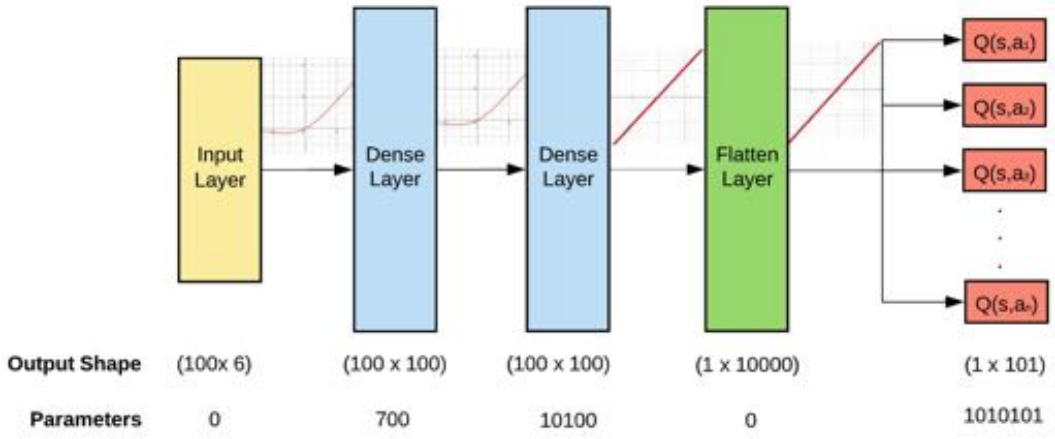


Figure 7 - DQN Architecture – Approach 2

4.2.2. Training strategy

The interaction of the Deep Q-Learning model with the scheduling environment is explained in more detail in this subsection and summarised in Figure 8.

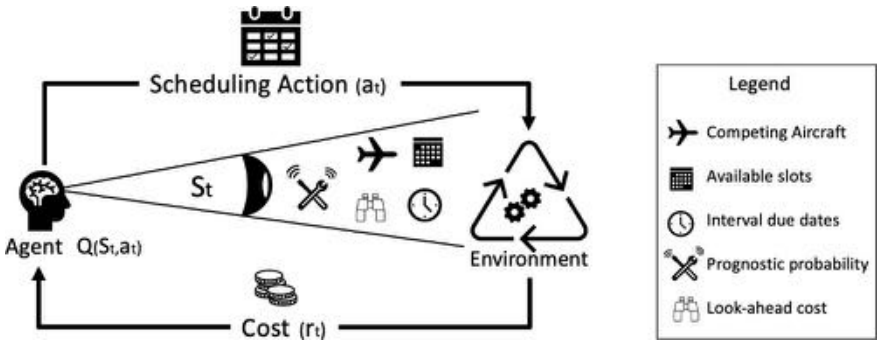


Figure 8 - Agent-environment interaction - Approach 2

The classic reinforcement learning feedback loop is tailored to the AMSP problem using the process shown in Figure 9. Initially, the environment is reset with the last maintenance execution dates and the complete list of tasks and routine blocks that form part of the aircraft maintenance program. The due dates of each aircraft are calculated based on the incremental utilisation that is simulated at each timestep, and the scheduling unit with the earliest due date is selected as the most urgent one.

In the following step, the state vector is calculated and fed as input to the DQL model. The DQN outputs a series of Q-values representing the discounted cost sum of the available discretised actions to the agent. The best action is selected by taking the minimum Q-value in order to minimise the cost function. Furthermore, in the case that the model is processing a routine block, the scheduling unit is directly allocated. Alternatively, when a CBM task is considered, an additional loop is introduced in the agent-environment interaction that updates the state based on a new RUL prognostic. In this way, it is possible to synchronise the DQN decision model with the Gaussian propagation of the prognostic uncertainty. Once the scheduling unit is allocated, the transition to

the next state occurs, and the DQN update loop initiates. The blocks highlighted in red indicate the steps required to calibrate the neural network. Firstly, the target network is updated every ten steps in order to ensure learning stability. Then, the experience observed at every agent-environment interaction step is sampled from a finite buffer memory. The sampling occurs in batches of 32 steps, where the stored experiences are fed as a tuple, with the form $< s, a, r, s' >$, and the loss function is calculated in order to minimise the deviation between the online and the target networks. The episode loop concludes when the simulation horizon is reached, while the training loop terminates after a sequence of 100 episodes.

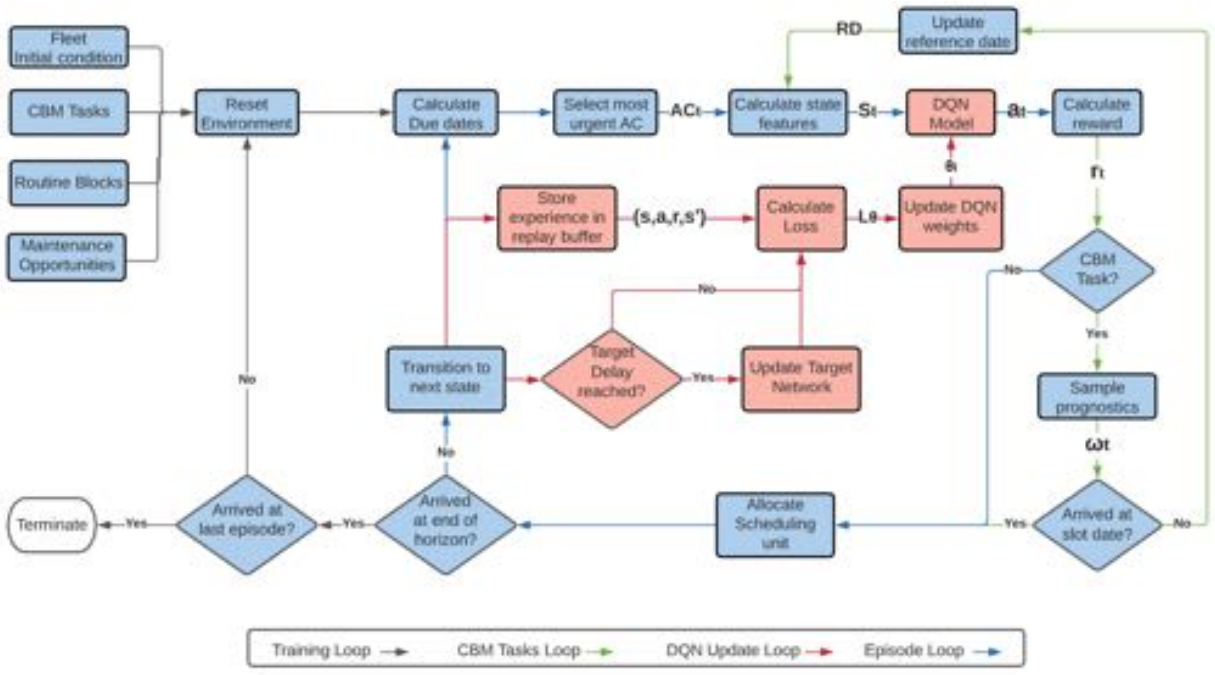


Figure 9 - Training strategy of the DQN algorithm - Approach 2

4.2.3. DQN configuration

In general, neural networks can be described as parametric functions that need to be calibrated during the training process. Nevertheless, some parameters are established beforehand in order to initialise the model. The scheduling units have been discretised in sub-intervals of 1% in order to have a fine mesh of all the possible slots. Consequently, the neural network input layer has a size of 100 neurons. The model is composed of three sequential hidden layers, comprising 100 neurons, each with a swish activation function. The activation function converts the input of the neurons with a non-linear transformation. Instead, the final layer is designed with a linear activation function because the output represents the next-state's Q-value and should converge to the discounted reward value.

Moreover, the output layer is sized with 101 neurons corresponding to the discretised sub-intervals and the AOG action. The learning rate, the exploration decay and the discount factor were selected upon a careful sensitivity analysis. The discount factor is an essential component as a very low value tends to create a greedy behaviour that compromises the future fleet opportunities. In contrast, a very high discount factor does not provide relevant information to the agent. Since, the maintenance scheduling problem is solved with a long-term horizon, the decisions far in the future are not relevant to actions taken at the beginning of the simulation. The optimal model performance is achieved when discounting future rewards by a 0.5 factor. Lastly, the Q-Learning model has been

trained over a period of 100 episodes in order to observe convergence. A complete list of the relevant DQN parameters is reported in the table below.

Table 3 - DQN hyperparameters - Approach 2

Parameter	Value
Learning rate (α)	0.0001
Discount factor (γ)	0.5
Initial exploration rate (E0)	1.0
Final exploration rate (ET)	0.01
Exploration rate decay (dE/dt)	0.9
Target delay (τ)	10
Batch size	32
Hidden layers	3
Dense size (neurons)	100
Training episodes (T)	100

4.2.4. Training performance

The training exercise consists of a narrow-body fleet maintenance schedule composed of 16 aircraft, with a planning horizon of 12 months. The slot opportunities are assumed to be available twice a week for a duration of 24 hours. Normally, these resources can only be used by a single aircraft, based on the airline policy that grounds each aircraft every 1500 flight hours. We assume that one slot can be used by more aircraft that share the labour hours capacity when the flight hour interval policy is reduced. Furthermore, the flight schedule data of the previous five years has been used to estimate the aircraft utilisation, and the last maintenance execution of each aircraft is used to initialise the problem. The resulting schedule is analysed in order to recommend the adaptive maintenance policy that maximises the benefits of a CBM strategy. The quality of the maintenance scheduling policy is evaluated by the interval utilisation, labour hours, and aircraft availability.

The DQN training performance is monitored to ensure that the agent can converge towards the optimal policy and minimise the cumulative cost function. In order to reduce the computational efforts required to train the DQL agent, the learning is performed across a reduced instance of eight months for each case. During the initial training phase, an exploration period takes place where sub-optimal decisions are selected multiple times, and the agent learns the impact of each action with respect to the state information. The training evolution is characterised by fluctuations in the cumulative rewards due to the DQN calibration process in a stochastic environment. The cumulative rewards are smoothed out using a simple moving average (SMA) over ten samples to assess the training performance. Figure 10 shows the SMA of the Deep Q-Learning algorithm with different scheduling policies. The vertical axis of the training evolution figure is scaled with a logarithmic transformation based on the shape of the reward function in order to capture the resolution of the rewards at each phase. The buffered areas around the SMA curve represent one standard deviation of the ten samples included in the calculation. The figure refers to four maintenance interval policies, as will be discussed in Section 5. From the figure, it is possible to visualise that the four policies converge to a steady-state value showing a learning behaviour over

the training evolution. One takeaway from the training observation is that the variation of the SMA, visible in the buffered areas, is larger when reducing the flight hours in the maintenance block cycle. This phenomenon occurs because there are more maintenance opportunities to explore, and the due date can be postponed more often based on the prognostics re-evaluation.

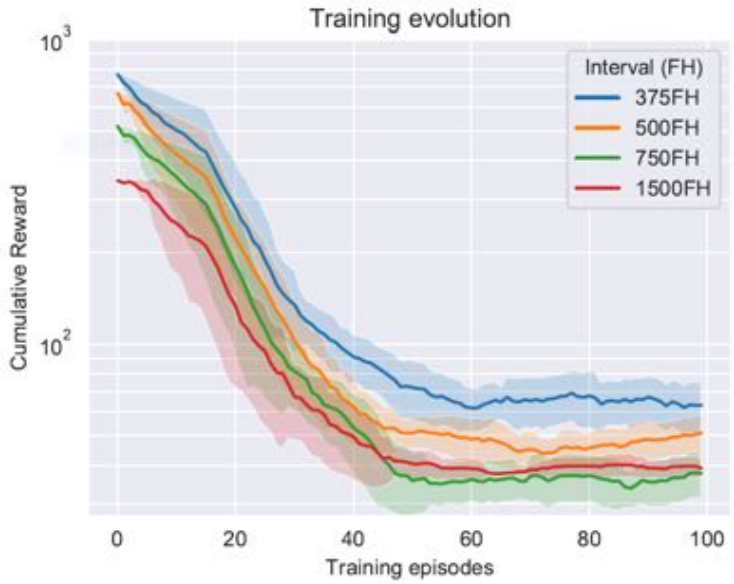


Figure 10 - SMA of the cumulative rewards over training episodes.

5. Benchmark case – Blocks interval policies

This section details a case study for segmenting the current maintenance interval policy into more frequent and lighter blocks. The increase in maintenance opportunities facilitates the integration of prognostics in the maintenance process. New blocks are obtained for each interval policy using a clustering MILP that minimises the repetition of tasks in the block cycle.

5.1. Case study

The input data used in this case study consist of 186 routine tasks belonging to the A-Check program followed by KLM Royal Dutch Airlines. The A-Check program comprises a total of 24 blocks spaced by 1500 Flight Hours (FHs), 600 Flight Cycles (FCs) and 120 Calendar Days (DYs). This 24-block maintenance program corresponds to a total of 2296 task executions per aircraft spanning over a time horizon of approximately eight years. The following information is provided per task:

- Task type
- Man-hours and skills involved in the execution of each task
- Intervals for each task, either in FHs, FCs, or DYs
- Zones and access panels per task

Moreover, the considered fleet comprises 16 Boeing 787, which usually are assigned to long haul routes. Consequently, the flight hour interval tends to be the most constraining factor for maintenance operations. As such, all intervals have been converted to FHs, by using historical information of aircraft utilisation over five years.

To support and integrate prognostics in an optimal manner and follow a CBM task-based strategy in the future, a flexible maintenance planning process is required. Towards this direction and to address the uncertainty included in the prognostics outputs, more maintenance opportunities are required. These additional maintenance opportunities can be provided by segmenting the current maintenance interval policy of 1500 FHs into more frequent but shorter in duration maintenance blocks. In this way, the maintenance planning process can be adjusted more effectively to the varying CBM tasks predictions by taking advantage of the more blocks opportunities, allowing allocating the CBM tasks optimally. An overview of the explored interval policies is presented in Table 4.

Table 4 - Interval policies explored

Interval Policy (FHs)	Number of Blocks	Ground time (hrs)
1500	24	24
750	48	12
500	72	8
375	96	6

5.2. Blocks generator

The benefit of clustering tasks in routine blocks is the reduction of the problem size. In fact, there is an analogy between the maintenance scheduling problem and a puzzle. The more pieces there are, the more complicated the problem becomes. There is no doubt that a highly segmented maintenance strategy could be more optimal from a mathematical perspective. However, scheduling maintenance implies a disturbance to operations because the aircraft has to be grounded for a certain period of time. Moreover, from a planning perspective, overhead costs related to hangar, engineers, and tool availability must also be considered. In recent years, airlines have begun to explore policies that increase the number of checks and reduce the ground time. This enables overnight checks and increases aircraft availability (Muchiri and Smit, 2009). These strategies foster the integration of CBM in the airline industry. In fact, if the aircraft is grounded more times for shorter periods, there are more maintenance opportunities that could be used to perform individual tasks monitored by prognostics. To address this hypothesis, we consider different maintenance block cycle policies that vary the frequency of the routine blocks and the respective maintenance elapsed time.

Figure 11 shows the logic behind different maintenance block cycle policies and the clustering algorithm. The cycle on the left considers a maintenance policy with 1500 flight hours between every check, while the cycle on the right employs a policy spaced by 750 flight hours. Therefore, twice as many blocks appear in the same horizon when the maintenance frequency is increased. Nevertheless, the duration of these routine blocks is halved to maintain a similar amount of labour hours between the two policies. Additionally, two other maintenance policies with 500 and 375 flight hours between blocks are evaluated to observe scheduling performance in such scenarios. A calendar-based estimation for these policies would lead to the grounding of aircraft every three months (1500FH) to every three weeks (375FH). The implementation of these policies requires an adaptation of the available slots schedule, used as input for the RL model. The available slots are subdivided into multiple opportunities when maintenance frequency is increased. Thus, a 50% decrease in interval flight hours between the blocks (from 1500FH to 750FH) leads to creating two maintenance slots with half of the original labour capacity. One slot becomes a grounding opportunity for two aircraft. Similarly, when the flight hours are reduced to 500FH and 375FH, one maintenance slot in the original slot calendar becomes a slot for 3 and 4 aircraft, respectively.

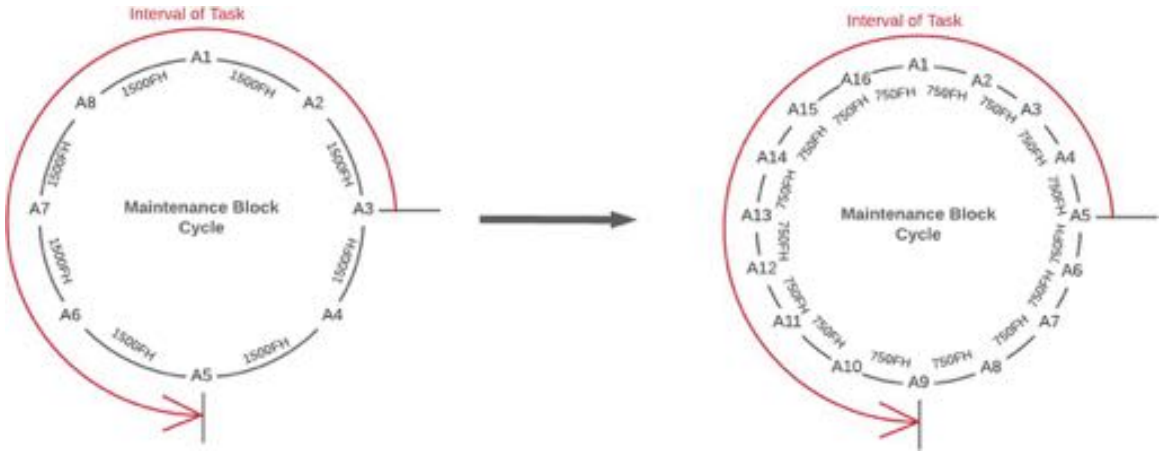


Figure 11 - Clustering strategy

The traditional maintenance policy of the reference airline clusters all the routine tasks with a 1500FH policy. Their task-packaging strategy involves several expert criteria that are used to produce an operational maintenance block cycle. Among others, the interval requirement, the labour hours, the skill of the licensed personnel, the aircraft zone, the inventory items, and other task inter-

dependencies are considered in the task-packaging process (Pereira and Ashok Babu, 2016; Ozkol and Senturk, 2017). There are two main reasons for which the clustering algorithm is employed to create new maintenance task packages or routine blocks. Firstly, different maintenance block policies than the one proposed by the reference airline are explored, for which the interval between blocks is varied. Secondly, a portion of the routine tasks is removed and monitored individually using synthetic prognostics. Therefore, the composition of routine blocks changes and a new clustering methodology is required. For simplicity, tasks have been clustered only based on interval and slot capacity considerations in this research. The following parameters have been defined to formulate the clustering MILP.

$$\boldsymbol{minimize} \quad \sum_{j \in J} \sum_{i \in B} x_{ij}$$

$$\sum_{i \in B} x_{ij} \geq 1 \quad \forall\, j \in J$$

$$x_{ij} - \sum_{p \in P_{ij}} x_{pj} \leq 0 \quad \forall\, i \in B, j \in J$$

$$\sum_{j \in J} x_{ij} \cdot D_j \leq LH \quad \forall\, i \in B$$

The first equation presents the objective function of the clustering algorithm. The objective is to minimise the number of times a task is repeated in the maintenance block cycle, where x_{ij} is the binary decision variable that defines if task j is assigned to block i, J is the set of routine tasks for an aircraft, and B is the set of blocks based on the maintenance policy configuration. The coverage constraints, presented in the second equation ensure that all the maintenance tasks are selected at least once in the maintenance block cycle. The third equation is a set of constraints that prevents a task from being assigned to a block past its interval limitation. In these constraints, P_{ij} is the set of blocks that have a distance from block i that is equal or less than the interval of the task j. The equation makes sure that if task j is allocated in block i, it appears one or more times in the previous blocks P_{ij}. The reader is referred to the policy on the left of Figure 11 for a graphical definition of set P_{ij}. If the model considers placing a task in block A3, the same task should appear at least once in the blocks covered by the red line in order to respect the interval limitation. In the depicted case, Pij is composed of blocks A5, A6, A7, A8, A1, and A2. The last constraints of the MILP model ensure that the number of tasks allocated in a certain block is constrained by the maximum labor hour capacity (LH).

5.3. Output

The output of the task packaging approach consists of a set of blocks together with the assigned tasks, as determined by the clustering MILP and respecting the constraints described in the previous section. The number of blocks is defined according to the selected interval policy, presented in Table 5, resulting in a total of 4 different outputs, one for each interval policy.

The output is obtained in the form of a .xlsx file, including the task name, the block number, and the interval of each task. A screenshot of the output in Table 5 (task names are anonymised for confidentiality reasons).

Table 5 - Example of the output generated

Task Name *	Block Number	Frequency (FHs)
xxx_]TTMF\[KKKX	A01_Block	1500
xxxFFIZEI]X`[Z`	A01_Block	1500
xxx`RIWL_]XGFDI	A01_Block	1500
xxxSCSQHCYCYMJD	A01_Block	2000
xxxFBGGEDHB[O^^	A01_Block	2000

* Task names anonymised.

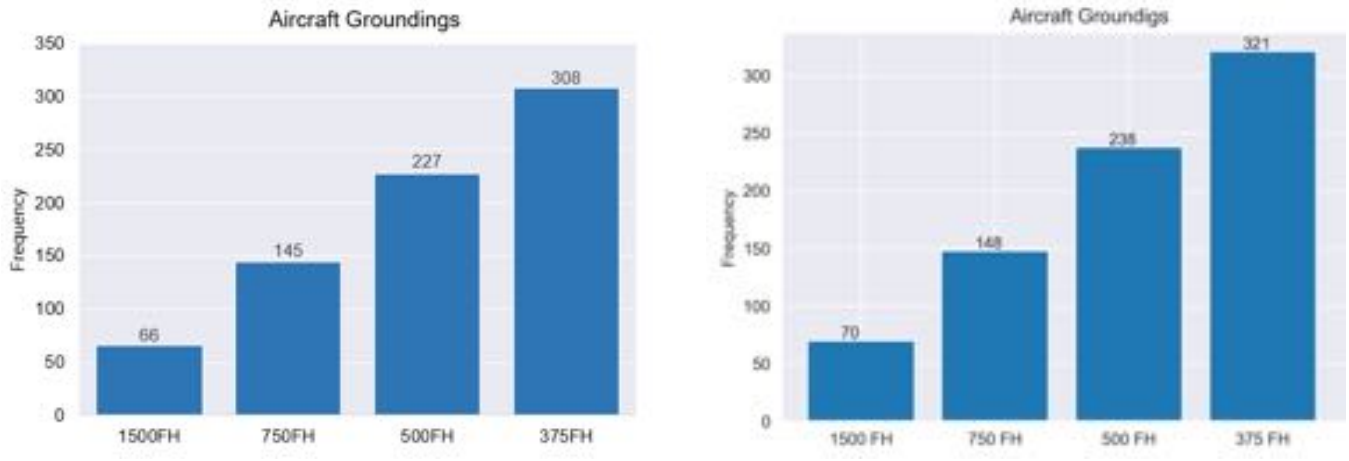
6. Algorithm comparison – Block interval policies

The scheduling algorithms detailed in this document were applied in studying different block cycle policies. The idea is to verify the quality of maintenance plans as the number of hangar visits for each aircraft increases. For simplicity, we will refer to the comprehensive approach from Section 3 as the first algorithm and the state space discretisation approach from Section 4 as the second algorithm. Both algorithms use the blocks obtained with the clustering method from Section 5 for each policy and have the same horizon. We assume maintenance can be performed twice a week, on Tuesdays and Thursdays. The available slots depend on the followed policy and vary from one 24h slot per day on the 1500FH case to four 6h slots on the 375FH case.

The solutions are evaluated with the following Key Performance Indicators (KPIs):

- Number of blocks scheduled for the fleet
- Total maintenance hours for the fleet
- Blocks interval utilisation

The total maintenance blocks scheduled represent the number of aircraft groundings during the defined horizon which directly impacts fleet availability. The results indicate that the number of groundings in each policy is identical for both algorithms, as shown in Figure 12. Nevertheless, the second algorithm seems to produce plans with approximately 4% more aircraft groundings for the considered horizon. Consequently, the total maintenance hours scheduled for the fleet also follow the same trend (Figure 13).



(a) (b)

Figure 12: Aircraft groundings per policy (a) for the first algorithm and (b) for the second algorithm.

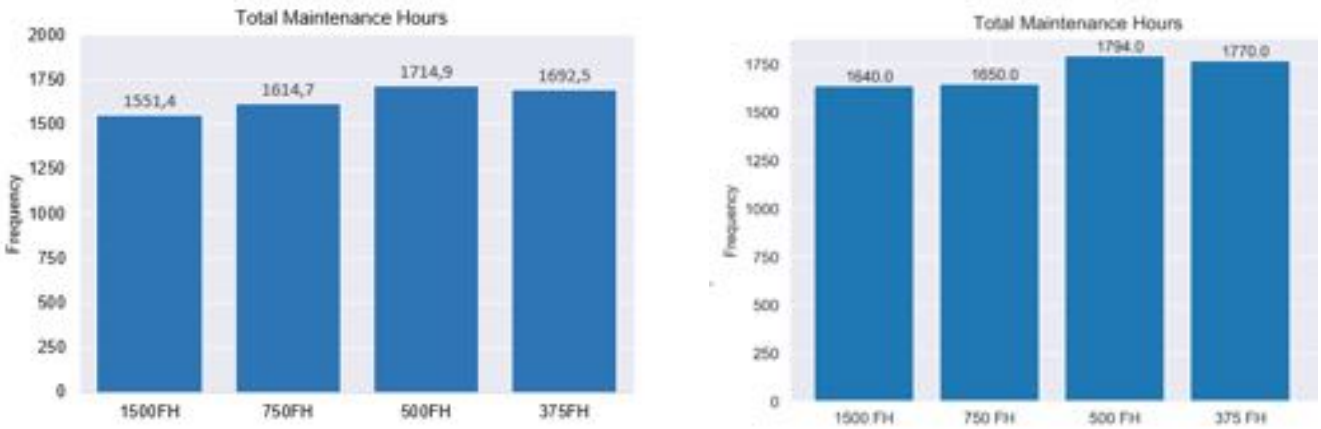


Figure 13: Total maintenance hours per policy (a) for the first algorithm and (b) for the second algorithm.

Results regarding block interval utilisations are presented in Figures 14 and 15, respectively, for the first and second algorithms. Both produce plans with similar block utilisation distributions for all policies. The main difference between them is on the two higher interval frequencies. The first algorithm produces better results for a 1500FH policy, while the second one is better for a 750FH policy. However, the trend of both algorithms is for the average block utilisation to decrease with the lower interval policies of 500 and 375 flight hours.

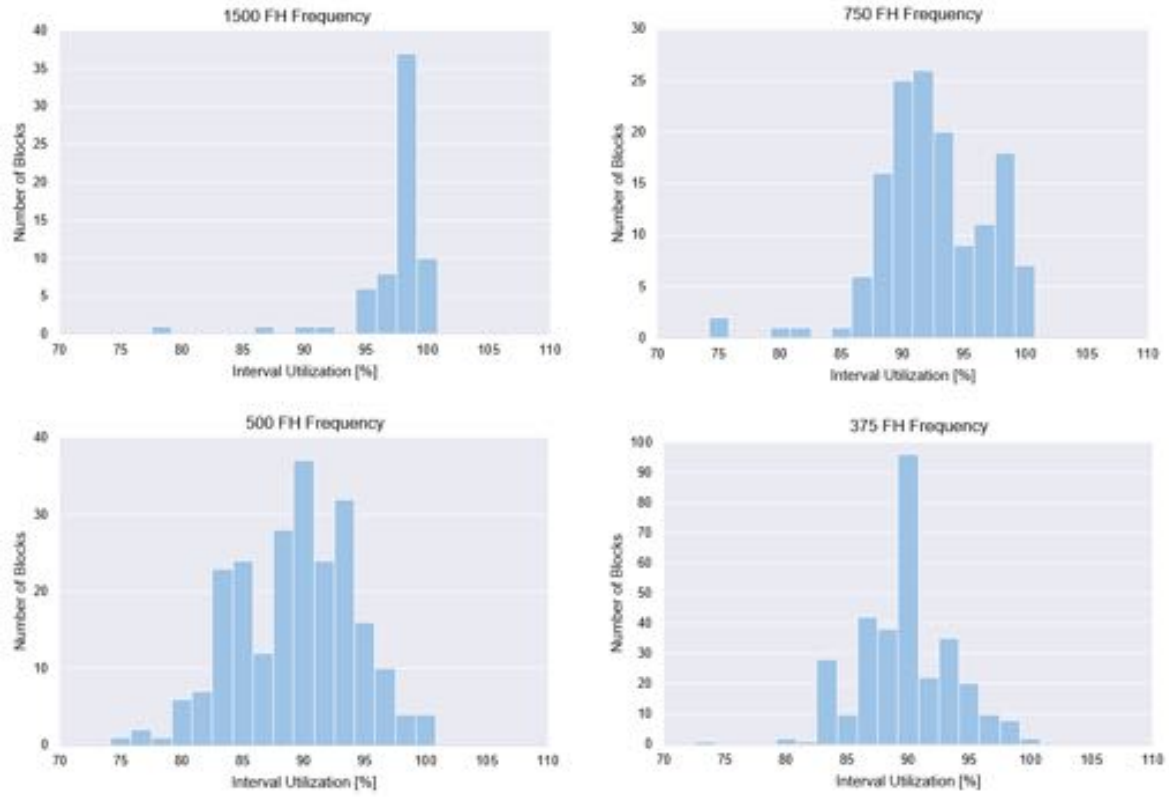


Figure 14: Block interval utilisations per policy for the first algorithm.

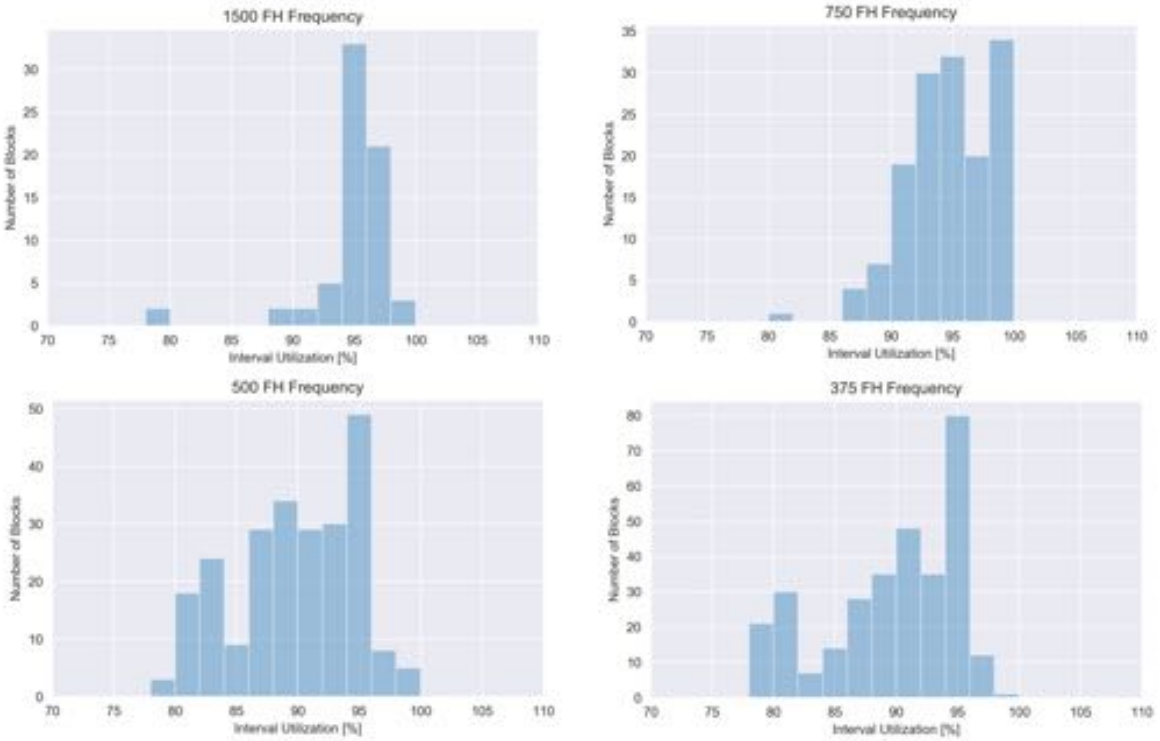


Figure 15: Block interval utilisations per policy for the second algorithm

7. Conclusion

This document described the work under task 6.3 of ReMAP's WP6 concerning developing a reinforcement learning algorithm for maintenance planning. Two approaches were developed, both using the Deep Q-learning algorithm to produce optimised maintenance plans. The first one consists of a comprehensive method for scheduling routine maintenance tasks. It has a check scheduling phase, where the fleet maintenance slots are defined, and a task packaging phase, where routine tasks are packaged individually into the predefined slots. The second approach was developed following task 6.2 and combined the scheduling of routine tasks and predictive-based tasks. This approach assumes routine tasks are allocated in blocks, and predictive tasks with stochastic due dates are scheduled individually. A case study for increasing the frequency of A-blocks is also reported. The idea is to support the integration of prognostics by having more maintenance opportunities and increasing the flexibility of the maintenance plan. To compensate for the frequency increase, new blocks require less ground time. These new blocks are created using a clustering MILP that minimises the number of times tasks are repeated in the maintenance cycle. Both scheduling algorithms were compared using these new blocks as input and continued producing good results for lower interval policies.

7.1. Lessons learned

The work developed under this task allowed us to acknowledge the following points:

- RL proved to be a good solution to solve the maintenance scheduling problem by fast producing quality plans. It also provides the ability to adapt to new maintenance conditions without needing to retrain the model is a relevant factor.
- While scheduling routine tasks individually might be more efficient regarding interval utilisation, it is not always the best strategy to apply in practice. Having fixed and predefined routine blocks is usually an easier approach to deal with inventory problems and workforce planning.

8. Bibliography

Deng, Q., Santos, B. F., and Curran, R. (2020). A practical dynamic programming based methodology for aircraft maintenance check scheduling optimisation. European Journal of Operational Research, 281(2):256–273.DOI: https://doi.org/10.1016/j.ejor.2019.08.025

Hasselt, H. V., Guez, A., and Silver, D. (2016). Deep Reinforcement Learning with Double Q-Learning. In AAAI Conference on Artificial Intelligence. URL: https://www.aaai.org/ocs/index.php/AAAI/AAAI16/paper/download/12389/11847.

Kingma, D.P.; Ba, J. (2014). Adam: A Method for Stochastic Optimisation. URL: https://arxiv.org/abs/1412.6980

Lagos, C., Delgado, F., and Klapp, M. A. (2020). Dynamic Optimisation for Airline Maintenance Operations. Transportation Science, 54:998–1015.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S., and Hassabis, D. (2015). Human-level control through deep reinforcement learning. Nature, 518(7540):529–533.

Muchiri, A. and Smit, K. G. (2009). Application of Maintenance Interval De-Escalation in Base Maintenance Planning Optimization. Enterprise Risk Management, 1(2). DOI: https://doi.org/10.5296/erm.v1i2.179

Ozkol, I. and Senturk, C. (2017). The effects of the use of single task-oriented maintenance concept and more accurate letter check alternatives on the reduction of scheduled maintenance downtime of aircraft. In 2017 8th International Conference on Mechanical and Aerospace Engineering (ICMAE), pages 67–74.

Pereira, M. A. and Ashok Babu, J. (2016). Information Support Tool for Aircraft Maintenance Task Planning. International Advanced Research Journal in Science, Engineering and Technology, 3(2).

Powell, W. B. (2011). Approximate dynamic programming: solving the curses of dimensionality. Wiley.

Powell, W. B. and Topaloglu, H. (2006). Approximate Dynamic Programming for Large-Scale Resource Allocation Problems. In Models, Methods, and Applications for Innovative Decision Making, pages 123–147. INFORMS.

Sutton, S. R., and Barto G. A. (2018). Reinforcement Learning: An Introduction. MIT Press.

Ramachandran, P., Zoph, B., and Le, Q. V. (2017). Searching for Activation Functions. URL: https://arxiv.org/abs/1710.05941

Watkins, C., and Dayan, P. (1992). Q-learning. Machine Learning. vol. *8*, pages 279–292.