Normal Paper ☒
Student Paper ☐
Young Engineer Paper ☐

Aircraft fleet readiness optimisation using reinforcement learning: a proof of concept

Kilian Vos [1], Zhongxiao Peng [1] and Wenyi Wang [2]

[1] *University of New South Wales, Sydney, NSW, 2052, Australia*
[2] *Defence Science and Technology Group, Fishermans Bend, VIC, 3207, Australia*

Abstract

A fleet of aircraft can be seen as a set of degrading systems that undergo variable loads as they fly missions and require maintenance throughout their lifetime. Optimal fleet management aims to maximise fleet availability and readiness while minimising overall maintenance costs. To achieve this goal, individual aircraft, with variable age and degradation paths, need to operate cooperatively to maintain high fleet availability while avoiding mechanical failure by scheduling preventive maintenance actions. Thereby, fleet management is a complex decision-making problem. In recent years, Reinforcement Learning (RL) has emerged as an effective method to optimise sequential decision-making problems (e.g., DeepMind's AlphaZero). In this work, we introduce an RL framework that can be employed to optimise the operation and maintenance of a fleet of aircraft. The operation of a fleet of aircraft is modelled in a simulated environment and Q-learning is employed to find the optimal policy. The RL solution is then evaluated against traditional operation/maintenance strategies and the results indicate that the RL policy performs relatively well over the fleet's lifetime. We conclude that RL has potential to help optimise and support fleet management problems.

Keywords: reinforcement learning, markov decision process, fleet operation, maintenance scheduling, readiness optimisation.

Introduction

Reinforcement Learning (RL) is a machine learning technique suitable to solve sequential decision-making problems (e.g., autonomous driving, inventory management or chess artificial intelligence). It distinguishes itself from supervised learning in many ways. While in supervised learning we provide examples to our models from which they can learn how to match the inputs to exact outputs (labels or numeric values), when dealing with a sequential decision-making problem it becomes very challenging to provide explicit supervision. Instead, in the RL framework we provide the algorithms with a reward function and let the algorithm explore the different actions and learn from experience [1].

Previous studies have investigated the applicability of RL to optimise the operation and maintenance of a single mechanical component [2], the maintenance schedule for a multi-component system [3], as well as the maintenance of a fleet of aircraft [4]. In this work, we present a proof of concept that uses RL to optimise both the operation (mission assignment) and maintenance schedule of a fleet of aircraft.

Degradation Model

The degradation model used to simulate aircraft damage was designed to follow Paris' Law and replicate the degradation paths observed in the Virkler experiment [5]. The crack length propagation is described by the following equation:

$$x(t+\Delta t) = f_0\,\Delta t\, C\,\left(\Delta_\sigma \sqrt{\frac{\pi\, x(t)}{\cos\frac{\pi\, x(t)}{2b}}}\right)^m \tag{1}$$

where $x(t+\Delta t)$ is the new crack length after a period Δt, f_0 the cyclic stress frequency (in Hz), C the Paris coefficient, m the Paris exponent, Δ_σ the stress range (in MPa), b the body width (in metres) and $x(t)$ the initial crack length (in metres). Each tail number in the fleet is assigned a different C coefficient drawn from a normal distribution (with median 8.586e-11 and standard deviation 0.619e-11). The Paris exponent m is set to 2.9. With these parameters the degradation model is capable of reproducing degradation paths that are similar to the ones observed during the Virkler experiment [5], based on 68 replicate crack propagation tests of aluminium alloy panels under constant load and environmental conditions. In the Virkler experiment, the initial crack length was 9 mm and the body width 76.2 mm, the tests were stopped when the crack length reached 50 mm. Figure 1 shows the similarity between the experimental degradation paths in the Virkler dataset (Fig. 1a) and degradation model described above (Fig. 1b).

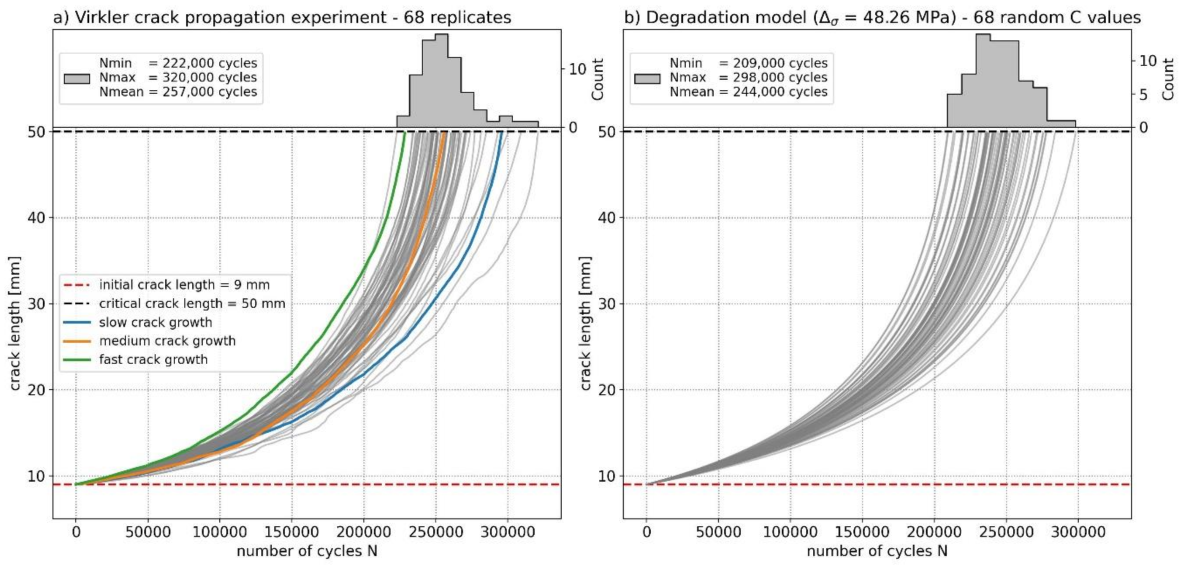

Figure 1. ***a)*** *Degradation paths for the 68 replicates in the Virkler fatigue crack propagation experiment* [5]. ***b)*** *Degradation model as described in Eq. 1) using a stress range Δ_σ = 48.26 MPa and Paris exponent m = 2.9. The degradation paths are plotted for 68 Paris coefficients (C in Eq. 1) drawn from a normal distribution.*

Aircraft Manoeuvres and Missions

While the Paris coefficients (C values) are drawn randomly for each aircraft in the fleet and then used throughout their lifetime, the stress range Δ_σ varies depending on the manoeuvre that the aircraft performs. The 5 manoeuvres that an aircraft can perform are described in Table 1. Mission types were designed by combining different manoeuvres carried out for a certain length of time. The composition of the 5 mission types (M1 to M5) is reported in Table 2. For example, Mission 1 consists of 2600 seconds of level fly (mv1), 60 seconds for take-off and landing (mv2), 30 seconds of barrel rotate (mv3) and 10 seconds of vertical up/down (mv5). A cyclic stress frequency f_0 of 5 Hz is used to propagate the crack length for the duration of each manoeuvre. Eq. (1) can now be used to calculate the damage (i.e., crack growth) resulting from a tail number flying one of the missions.

Table 1. Manoeuvres and Missions

Manoeuvre Name	Short name	Stress range Δ_σ
Level Fly	mv1	30 MPa

Take-off/Landing	mv2	50 MPa
Barrel rotate	mv3	70 MPa
Pull up/Pull down	mv4	90 MPa
Vertical up/down	mv5	110 MPa

Table 2. Mission types

Mission types	
Name	Combination of manoeuvres
M1	mv1(2600s) + mv2(60s) + mv3(30s) + mv4(0s) + mv5(10s)
M2	mv1(3000s) + mv2(60s) + mv3(0s) + mv4(70s) + mv5(40s)
M3	mv1(3600s) + mv2(60s) + mv3(40s) + mv4(60s) + mv5(20s)
M4	mv1(4000s) + mv2(60s) + mv3(60s) + mv4(80s) + mv5(0s)
M5	mv1(4500s) + mv2(60s) + mv3(0s) + mv4(60s) + mv5(80s)

A set of missions is prescribed daily to the fleet, and each tail number can either fly one of the prescribed missions, stand-by or go under preventive maintenance. The goal is to maximise the fleet readiness and achieve the highest mission completion rate over the lifetime of the fleet. In Reinforcement Learning (RL), the mathematical formulation to solve a sequential decision-making problem is referred to as a Markov Decision Process (MDP).

Markov Decision Process: States, Actions and Rewards

An MDP is the formalism in which RL problems are described. As defined in [1], an MDP is a tuple with 5 variables *($\vec{S}$, $\vec{A}$, P_{sa} , R, γ)* where:

- $\vec{S}$ is a set of *states* that we observe (e.g., crack length or damage level).
- $\vec{A}$ is a set of *actions* (e.g., fly a mission, stand-by or apply preventive maintenance).
- P_{sa} are the *state transition probabilities,* which give the probability of transitioning to each other state when taking action *a* in state *s*.
- R is the *reward function* which maps each state-action pair to a reward value.
- γ is the *discount factor*, a value between 0 and 1 used to discount future rewards.

In the dynamics of an MDP, we start in a certain state s_0, choose action a_0, randomly transition to the next state s_1 and receive reward $R(s_0, a_0)$, then choose another action a_1 and transition to state s_2 and receive reward $R(s_1, a_1)$, and so on and so forth:

$$s_0 \xrightarrow[R(s_0,a_0)]{a_0} s_1 \xrightarrow[R(s_1,a_1)]{a_1} s_2 \xrightarrow[R(s_2,a_2)]{a_2} \ . \ . \ . \qquad (2)$$

The goal of RL algorithms is to choose the sequence of actions that maximise the expected value of the total payoff, which is the immediate reward plus the sum of future discounted rewards:

$$E[\, R(s_0, a_0) + \gamma R(s_1, a_1) + \ \gamma^2 R(s_2, a_2) + . \ . \ .] \qquad (3)$$

The function that maps the states to the actions $\pi: \vec{S} \rightarrow \vec{A}$ is referred to as a *policy*. While there are no universal methods for searching the optimal policy, popular approaches include dynamic programming (Value Iteration and Policy Iteration), Monte-Carlo methods and Time-difference methods (e.g., Q-learning) [1]. While some methods like dynamic programming require a model of the environment (i.e., known state transition probabilities *Psa*), other methods are model-free and do not require prior knowledge of the state transition probabilities. In this case, we do not know the state transition probabilities, so we decided to use Q-learning, a model-free method.

Q-learning

Q-learning is one of the most popular model-free algorithms to find an optimal policy in a finite MDP. The agent explores the state-action pairs in the environment and calculates the expected rewards for an action taken in a given state using Bellman's update [1]:

$$Q(\overrightarrow{S_t},\overrightarrow{A_t}) \leftarrow Q(\overrightarrow{S_t},\overrightarrow{A_t}) + \alpha\left[R_{t+1} + \gamma\, max_a Q(\overrightarrow{S_{t+1}},a) - Q(\overrightarrow{S_t},\overrightarrow{A_t})\right] \qquad (4)$$

where $Q(\overrightarrow{S_t},\overrightarrow{A_t})$ represents the expected sum of future rewards for taking action $\overrightarrow{A_t}$ in state $\overrightarrow{S_t}$ (known as the Q-value or quality of the state action pair), α is the learning rate and γ the discount factor. In Q-learning, the Q-values for each state-action pair encountered are stored in a look-up table and updated throughout the search. The search is usually conducted according to the epsilon-greedy strategy, where initially the agent explores the environment (by choosing random actions) and as it gains knowledge of the environment it gradually reduces the exploration rate and begins to exploit its knowledge (by choosing the action with highest Q-value).

A simple example with a fleet of 2 aircraft

A simple environment with a fleet of 2 aircraft and 2 different types of mission per day was first designed to test the RL methodology. The failure threshold was set to half of the body width (38.1 mm) and the Paris coefficients were chosen so that tail #1 has a slow degradation fast and tail #2 a faster one. Two missions are assigned each day, one M1 and one M5 (see Table 2), respectively the least and most damaging mission types.

The *State* vector ($\overrightarrow{S_t}$) tracks the crack length of each tail number in the fleet. To make it a discrete MDP, the crack length, a continuous variable between 9 and 38 mm, was discretised into 10 equally-spaced bins, hereafter referred to as damage levels. The *Action* vector ($\overrightarrow{A_t}$) contains the action chosen by the RL agent at timestep t.

$\overrightarrow{S_t} = \begin{bmatrix} d_{t,1} & d_{t,2} \end{bmatrix}$, where $d_{t,i}$ is the damage level of tail number *i* at timestep *t* (5)

$\overrightarrow{A_t} = \begin{bmatrix} a_{t,1} & a_{t,2} \end{bmatrix}$, where $a_{t,i}$ is the action assigned to tail number *i* at timestep *t* (6)

In this environment, there are 4 possible actions for each tail number: 1) Fly mission M1; 2) Fly mission M5; 3) Stand-by; 4) Preventive maintenance. This means that at each timestep the RL agent must choose one from 16 possible actions (4^2).

The *State transitions* define what happens when an action is chosen in a given state: i) when a tail number flies a mission, its damage level is propagated using Eq. (1); ii) when in stand-by, the damage level remains unchanged; iii) when preventive maintenance is chosen, the tail number's damage level is restored to its initial value (crack length of 9 mm) and it becomes unavailable for 5 days; iv) if the damage level goes above the failure threshold, the tail number is sent under corrective maintenance, its damage level is restored, it becomes unavailable for 5 days, but the agent incurs a higher cost than preventive maintenance.

The *Rewards function* help the agent achieve its goal and are defined as follows: i) R = +1 for flying M1; ii) R = +2 for flying M5; iii) R = -1 for stand-by; iv) R = -10 for preventive maintenance; v) R = -100 for corrective maintenance. This reward scheme encourages the agent to fly missions and send the aircraft under preventive maintenance before they reach the failure threshold.

The policy search is performed by exploring the environment over many episodes and creating a Q-table that stores and updates the Q-value of each state-action pair encountered according to Bellman's update (Eq. 4). To ensure that the environment is thoroughly explored (i.e., most possible states are visited) we use an *epsilon-greedy search strategy,* where both the learning rate α and the exploration rate ε decay as the search progresses. The decay curves for both parameters over 2,000 episodes are shown in Figure 2a. The exploration rate is the probability of choosing a random action over the action with the highest Q-value, while the learning rate dictates how much new observations impact the current Q-values (see Eq. 4). Reducing the learning rate guarantees that the policy converges, as shown in Figure 2b. Additionally, each episode is initialised in a different state, with the initial crack length sampled from a uniform distribution $U[9\ mm, 38.1\ mm)$.

A graphical visualisation of the RL policy is presented in Figure 2c. At low damage levels (< 6), the RL policy sends both tail numbers to fly missions (M1 or M5). The preventive maintenance and stand-by actions are only used at high damage levels to prevent failure during operation. Otherwise, the policy always tries to fly missions with the least-damaged tail number, while sending the most damaged one under maintenance. Both tail numbers are sent under maintenance simultaneously only in one case, when both are at damage level 9.

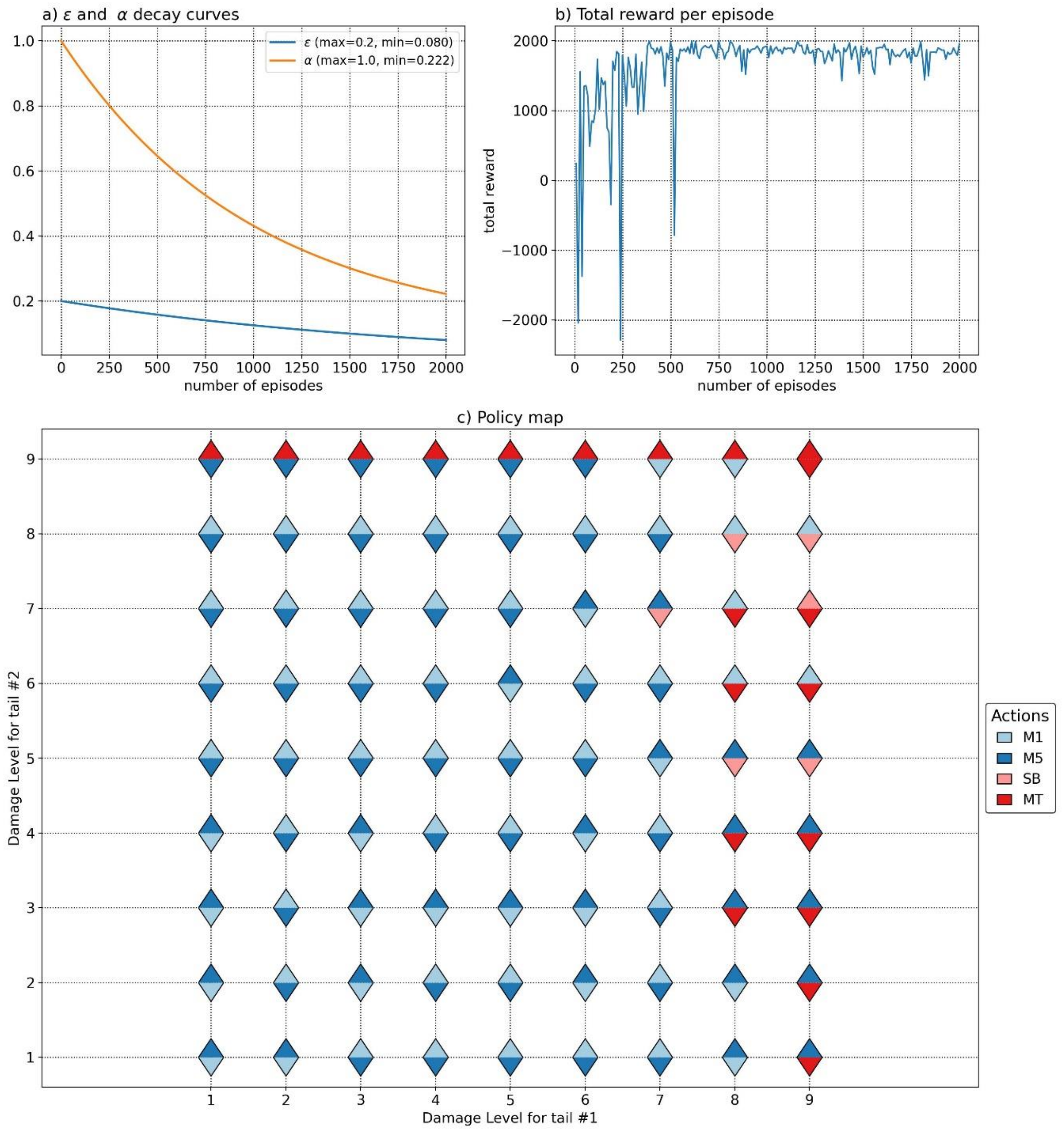

*Figure 2. **a)** Decay curves for the exploration rate (epsilon) and the learning rate (alpha). **b)** Evolution of the total reward achieved by the RL policy during the search. **c)** Policy map showing the preferred action in each possible state. The bottom triangle shows the action assigned to tail #1 while the top one is for tail #2.*

To evaluate the performance of the RL policy, a set of baseline policies were designed based on established fleet management methods:

a) *On-condition maintenance*: in this policy preventive maintenance is assigned whenever an aircraft's damage level goes above a fixed threshold. This threshold was optimised by trial-and-error, and it was found to provide the highest reward when set at a damage level of 8/10. Outside of the maintenance zone, this policy always flies the prescribed missions and assigns them randomly to each tail number.
b) *Force-life management*: this policy assigns the missions so that the cumulated damage (i.e., crack length growth) is kept equal for each tail number in the fleet. For example, if tail #1 has a higher cumulated damage than tail #2, then in the next timestep the most damaging mission is assigned to tail #2. Preventive maintenance is applied according to the *on-condition maintenance* strategy (when damage level is above 8).
c) *Equal*-stress policy: this strategy is like *force-life management* but with a small distinction. The *force-life management* strategy assumes that we have perfect knowledge of the cumulated damage for each tail number. However, in a more realistic scenario, we cannot record how much the crack propagated after each mission, but we do have knowledge of the difficulty of the missions that were flown by each tail number. Thereby, the *equal-stress* policy calculates the "theoretical" cumulated stress range of each tail number based on the missions it flew and uses that metric to assign the missions rather than the cumulated crack length. The cumulated stress range associated with flying each mission is calculated as $f_0 \sum \Delta t_i \, \Delta_{\sigma,i}$, where i refers to each manoeuvre in the mission as reported in Tables 1 and 2.

To evaluate the performance of each policy, 10,000 episodes were run with each episode initialised in a different state where the initial crack length of each tail number was sampled from a uniform distribution $U[9\,mm, 38.1\,mm)$. Figure 3 shows the distribution of the rewards for each policy. It results that the RL policy outperforms the 3 baseline policies between 4% (force-life management) and 10% (on-condition maintenance). This is a significant improvement and demonstrates the potential of RL to support fleet management operations.

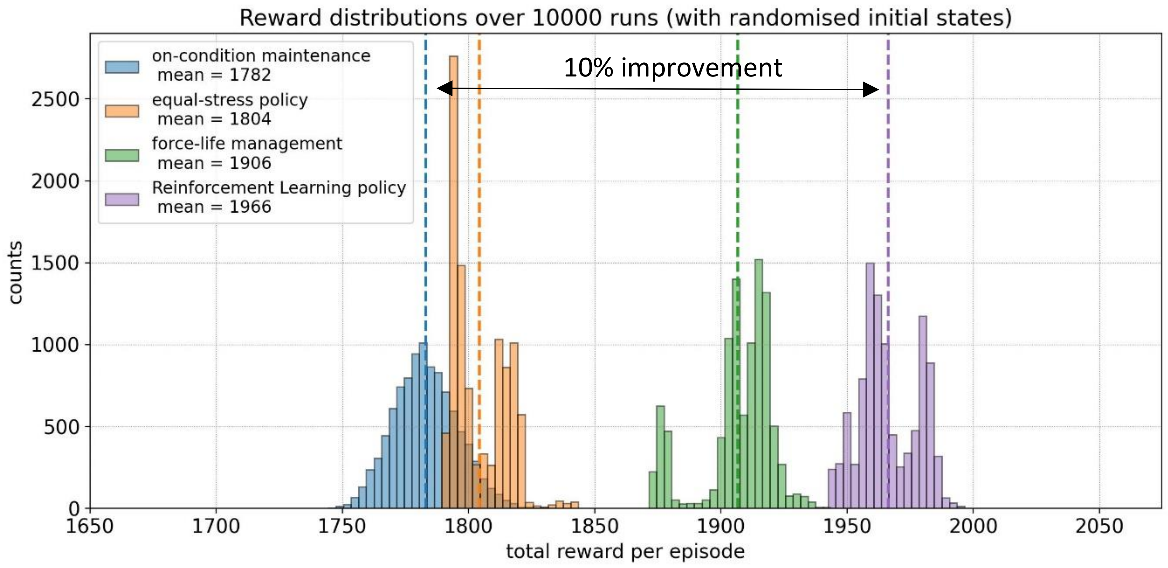

Figure 3. Evaluation of the Reinforcement Learning policy against baseline policies (on-condition maintenance, equal-stress and force-life management). The plot shows the distribution of the reward over 10,000 episodes initialised in randomised states.

Discussion on the scalability of this RL framework

While we were able to successfully optimise the Markov Decision Process with standard Q-learning in this small 2-aircraft fleet, in more realistic cases we would be dealing with larger fleets. However, standard Q-learning (using a look-up table) does not scale well for larger fleets. In fact, the dimensions of the state and action spaces increases exponentially with the number of aircraft and the types of missions according to the following relationships:

- $State\ space\ =\ 10^n$, where n is the number of aircraft in the fleet.
- $Action\ space\ =\ (\#\ mission\ types + 2)^n$, where the number of possible actions per tail number depends on the number of mission types (M1 to M5) included in addition to the stand-by and preventive maintenance actions.

For example, for the 2-aircraft fleet, we have 100 possible states (10^2) and 16 possible actions at each timestep (restricted to 2 mission types, so 4 possible actions per tail number), which results in a relatively small Q-table of 100x16. A larger environment with 6 tail numbers and the 5 mission types (M1 to M5) would lead to 10^6 possible states and choosing amongst 7^6 possible actions each timestep. With these dimensions, the Q-table is not computationally tractable. To overcome this limitation, Deep Q-learning has been proposed [1], a variant where the Q-table is replaced with a Neural Network that acts as a function approximator and learns the state-value pairs instead of storing them in the look-up table. For more realistic scenarios and larger fleets, future efforts should explore the applicability of Deep Q-learning.

Conclusion

This work has presented a proof of concept on the applicability of RL to support fleet management operations. Firstly, a synthetic environment was designed to simulate the degradation of mechanical components during the operation of the aircraft. The degradation model uses Paris' Law where the parameters were calibrated to reproduce the fatigue crack propagation observed in the Virkler experiment [5]. A set of missions are designed by combining different manoeuvres with varying degree of difficulty (i.e., varying stress range). This simulated environment can then be used to explore optimal strategies on how to manage a fleet of aircraft throughout their lifetime. At each timestep, a set of missions are prescribed to the fleet and the operator decides for each aircraft whether to fly a mission (positive reward), stand-by or send the aircraft to preventive maintenance (negative reward). The goal is to find the policy that maximises fleet availability and readiness while minimising overall maintenance costs. We demonstrate that Q-learning, a popular RL algorithm, can optimise this complex decision-making problem and produce a strategy that outperforms traditional fleet management strategies (e.g., on-condition maintenance and force-life management). Although our results are limited to a small computational space with a fleet composed of only 2 aircraft, they provide a valid framework that can be used to support fleet management operations. Future efforts should focus on designing more realistic fleet environments and explore the applicability of alternative RL algorithms that can optimise large computational spaces.

Acknowledgements

We thank A/Prof Lina Yao for her expert advice on RL and support during this project.

Code Availability

Code to reproduce the results presented here is available at https://github.com/kvos/RLfleet.

References

[1] R.S. Sutton, A.G. Barto, Reinforcement Learning, Second Edition | The MIT Press, 2015. https://mitpress.mit.edu/books/reinforcement-learning-second-edition (accessed March 2, 2021).

[2] L. Bellani, M. Compare, P. Baraldi, E. Zio, Towards Developing a Novel Framework for Practical PHM: a Sequential Decision Problem solved by Reinforcement Learning and Artificial Neural Networks, Int. J. Progn. Heal. Manag. 31 (2019) 1–15. https://www.researchgate.net/publication/339016560 (accessed March 2, 2021).

[3] N. Yousefi, S. Tsianikas, D.W. Coit, Dynamic maintenance model for a repairable multi-component system using deep reinforcement learning, Qual. Eng. 34 (2022) 16–

35. https://doi.org/10.1080/08982112.2021.1977950.

[4] V. Mattila, K. Virtanen, Scheduling fighter aircraft maintenance with reinforcement learning, in: Proc. 2011 Winter Simul. Conf., 2011: pp. 2535–2546. https://doi.org/10.1109/WSC.2011.6147962.

[5] D.A. Virkler, B.M. Hillberry, P.K. Goel, Statistical Nature of Fatigue Crack Propagation., Tech Rep AFFDL TR Air Force Flight Dyn Lab US TR-43-78. (1978).