Reinforcement Learning for Strategic Airport Slot Scheduling: Analysis of State Observations and Reward Designs

1st Anh Nguyen-Duy
Air Traffic Management Research Institute
Nanyang Technological University
Singapore
nguyendu002@e.ntu.edu.sg

2nd Duc-Thinh Pham
Air Traffic Management Research Institute
Nanyang Technological University
Singapore
dtpham@ntu.edu.sg

3rd Jian-Yi Lye
School of Computing and Information Systems
Singapore Management University
Singapore
jianyi.lye.2020@scis.smu.edu.sg

4th Duong Ta
School of Computing and Information Systems
Singapore Management University
Singapore
donta@smu.edu.sg

***Abstract*—Due to the NP-hard nature, the strategic airport slot scheduling problem is calling for exploring sub-optimal approaches, such as heuristics and learning-based approaches. Moreover, the continuous increase in air traffic demand requires approaches that can work well in new scenarios. While heuristics rely on a fixed set of rules, which limits the ability to explore new solutions, Reinforcement Learning offers a versatile framework to automate the search and generalize to unseen scenarios. Finding a suitable state observation and reward structure design is essential in using Reinforcement Learning. In this paper, we investigate the impact of providing the Reinforcement Learning agent with an intermediate positive signal in the reward structure along with the use of the Full State Observation and the Local State Observation. We perform training with different combinations of the reward structure, the state observation, and the Deep Q-Network (DQN) algorithm to define the training efficient formulation. We use two types of scenarios, medium and high-density, to test the ability to generalize to unseen data of the approach. Each type of scenario is used to train two separate models, Model 1 and Model 2. Model 1, which is trained on high-density scenarios, will be tested with medium-density scenarios; the results obtained will then be compared with the results of Model 2, and vice versa. We additionally analyze the performance of the DQN models with the Proximal Policy Optimization (PPO) models. Results suggest that combining the Local State Observation and the intermediate positive signal leads to a stable convergence. The obtained DQN models perform better compared to the PPO models, achieving an average displacement per request of 1.44/1.99 while only having on average 0.00/0.02 unaccommodated requests for medium/high-density scenarios. The t-statistic of 0.0810/-1.0016 and the p-value of 0.9356/0.3190 also suggest that the DQN models can generalize to unseen scenarios.**

***Index Terms*—Reinforcement Learning, airport slot scheduling, strategic**

I. Introduction

The significant increase in air traffic demand leads to severe airport congestion, which results in flight delays, economic loss, and environmental pollution [1], [2], [3]. While a supply-side solution, building more infrastructure, is infeasible in the short-term and capital intensive, a demand-side approach, airport slot scheduling or airport slot allocation, via effective control of the distribution of the demand can help to adapt to the current infrastructure [4]. Current approaches for airport slot scheduling vary from exact methods to sub-optimal approaches; however, due to the NP-hard nature of the problem which makes it intractable for exact methods, the literature is calling for more investigations of sub-optimal approaches, such as heuristics and learning-based approaches [5], [6], [7], [8], [9]. Reinforcement Learning (RL), a learning-based approach, is rising as a potential candidate for the problem. Heuristics rely on a fixed set of rules, which limits the possibility of finding new solutions [10], [11]. Reinforcement Learning, on the other hand, can automate the search and explore freely new strategies. Furthermore, learning-based approaches like Reinforcement Learning can generalize to unseen scenarios and use the new scenarios as inputs for improving performance, i.e. the ability to self-evolve [12], [13], [14]. Reinforcement Learning has been applied extensively in the Air Traffic Management (ATM) domain [15], [16], [17], [18]. In the pre-tactical phase, Reinforcement Learning has been used to adjust the flight time of flights or trajectories by imposing delay to cope with congestion [19], [20], [13], [21], [22], [23]. Few works consider using Reinforcement Learning for airport slot scheduling at the strategic phase. While pre-tactical slot reallocation is a reactive measure concerning up-to-date information, strategic slot scheduling is a long-term planning to maximize resource utilization and ensure regulatory compliance. The difference between airport slot scheduling at the strategic phase and airport slot reallocation at the pre-tactical phase is that at the pre-tactical phase, the flight time is displaced later via imposing delay, while at the strategic phase, the flight time can be displaced not only later but also earlier, i.e. move the flight time ahead to the original time. Furthermore, in airport slot scheduling at the strategic phase, there are constraints relating to a series of slots associated with a request that guarantee the schedule regularity over a whole scheduling season. However, for simplicity, in this paper, we have not considered these constraints. We refer to the term ”airport slot scheduling” for ”airport slot scheduling/allocation at the strategic phase” throughout the paper.

To fill in the research gap, we study the use of Reinforcement Learning for airport slot scheduling. We first formulate the airport slot scheduling problem as a Markov Decision Process (MDP). We consider the reward structure commonly used for the pre-tactical airport slot allocation problem [13], [21], [22], which consists of the delay and overload components. We argue that adding a positive reward signal after solving an unaccommodated request will lead to a more stable training convergence. To prove our claim, we train different models with two reward designs, Reward Design A, with the positive reward signal, and Reward Design B, without the positive reward signal. Along with the two reward designs, we also analyze the use of the full state observation and the local (partial) state observation to verify the effectiveness of each type of state observation. We consider the following metrics for performance evaluation, the total schedule delay, the maximum displacement across all requests, and the number of unaccommodated requests. To estimate the ability to generalize to unseen scenarios of the models, t-statistic and p-value are used. Further details will be explained later. Our main contribution can be summarized as follows:

- We provide a Reinforcement Learning formulation dedicated specifically to the airport slot scheduling problem at the strategic phase.
- We prove that using the reward structure deducted from the current Reinforcement Learning approaches for pre-tactical airport slot allocation is not efficient for the strategic airport slot scheduling problem. Adding a positive reward signal in the reward structure helps to achieve convergence in training.
- We provide a convergence analysis of different combinations of the two reward structures along with the full state observation and the local state observation. We also test the generalizing ability of the obtained model(s).

II. Learning Mechanism

A. Generated scenarios

Figure 1 shows the overall concept diagram throughout this study. To apply Reinforcement Learning, it is necessary to have a learning environment for the agent to interact with. A scenario, that consists of the airlines' requests and the airports' capacity, is first generated by the learning environment. Each airline's request consists of two movement requests, departure and arrival. An airline's request $r \in R$ is a tuple $(a_{dep}, t_{dep}, a_{arv}, t_{arv})$, where $a_{dep}, a_{arv} \in A$ and

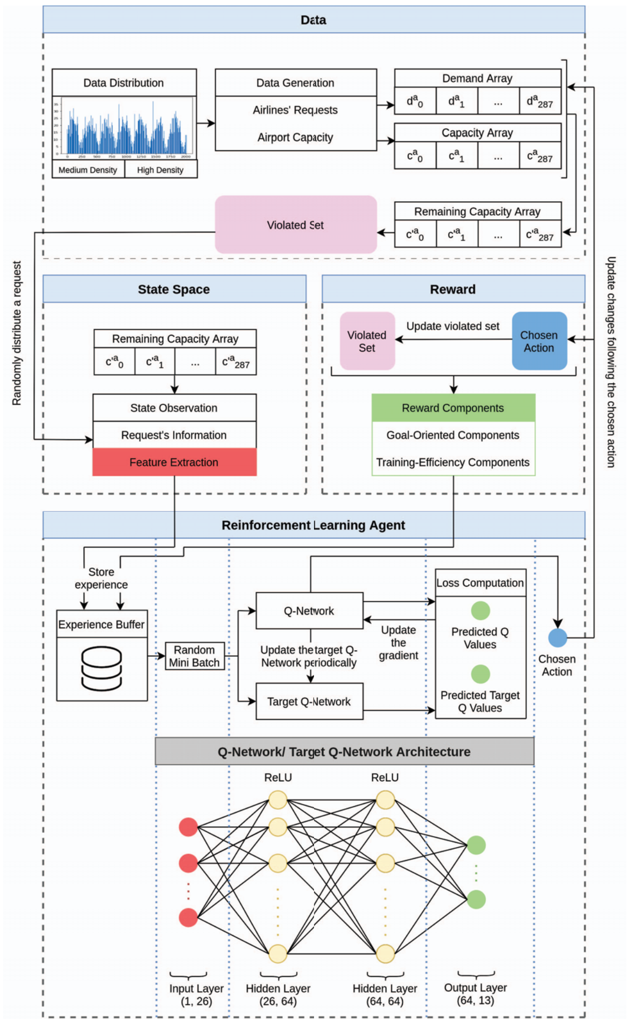


Fig. 1. The proposed framework for the learning mechanism.

$t_{dep}, t_{arv} \in T$. Each departure/arrival movement associated with a time slot $t \in T$ at airport $a \in A$ will add 1 unit to the current demand $d^a_t \in D^A_T$. The current demand d^a_t equals the total number of movements associated with the time slot t_a. The capacity of time slot t at airport a is denoted as c^a_t. Let $T_{violated} = \{t_a | d^a_t > c^a_t\}$ be the set of all violated time slots. If a request r has either t_{dep} or $t_{arv} \in T_{violated}$, the request is added to the violated set V. One of the RL agent's objectives is to clear all of the requests from V, i.e. assigning new time slots to the requests in V so that there is no more time slot t_a with $d^a_t > c^a_t$. It is important to note that clearing all requests from the violated set is not encoded as a hard constraint, but rather as a minimizing objective. Imposing it as a hard constraint is not flexible in considering the trade-off between the two conflicting objectives, minimizing the total schedule delay and minimizing the number of unaccommodated requests. Considering the maximum displacement per request is essential as it guarantees the schedule acceptability [24]. In our problem, we consider scenarios for two airports. We have two levels of density, medium and high density, to test the performance of the agent in generalizing to different unseen scenarios. Further details of the scenarios will be explained later. We provide below a summary of notations for the problem:

- $r = (a_{dep}, t_{dep}, a_{arv}, t_{arv}) \in R$: set of airlines' requests.
- $A = \{0, 1\}$: set of considered airports denoted by a, $a = 0$ encodes the first airport and $a = 1$ encodes the second airport
- $T = \{0, 1, ..., 287\}$: set of 5-minute-interval time slots t
- C_T^A: set of capacity constraint c_t^a of time slot t at airport a
- D_T^A: set of current demand d_t^a for time slot t at airport a
- $T_{violated} = \{t_a | d_t^a > c_t^a\}$: set of all violated time slots
- $V = \{r = (a_{dep}, t_{dep}, a_{arv}, t_{arv}) | t_{dep} \in T_{violated} \cup t_{arv} \in T_{violated}\}$: set of unaccommodated requests

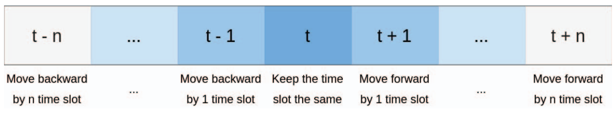
B. Learning environment and the agent interaction

At the beginning of each episode, the learning environment will generate a scenario with the airline's requests and the airports' capacity. We encapsulate the data in the form of arrays. All the arrays will be of the size of (1x288), where 288 is the number of 5-minute time slots in a day. Each airport a has its capacity array $Capacity\_Array^a = [c_0^a, c_1^a, ..., c_{287}^a]$ and current demand array $Current\_Demand\_Array^a = [d_0^a, d_1^a, ..., d_{287}^a]$. From the capacity and current demand arrays of each airport a, we obtain the remaining capacity array $Remaining\_Capacity\_Array^a = [\mathrm{c}'^a_0, \mathrm{c}'^a_1, ..., \mathrm{c}'^a_{287}]$, where $\mathrm{c}'_\mathrm{t} = \mathrm{c}_\mathrm{t} - \mathrm{d}_\mathrm{t}$. The remaining capacity arrays are used to identify the current unaccommodated requests. The mechanism of solving the unaccommodated requests one by one has been applied for heuristics [9], [25]. Therefore, we let the learning environment pick a request from the violated set and distribute it to the RL agent. The order of solving the unaccommodated requests may affect the final results if we consider a multiple-day scheduling horizon. Choosing requests with the highest number of spanning days over the scheduling horizon gives the best results [25]. However, in this paper, since we only consider a single-day scheduling horizon, there is no order of spanning days. A request from the violated set is, thus, randomly picked at each step. From the obtained request, the information of the request is encapsulated as state observations, the inputs for the RL agent to make decisions. After the agent takes an action, the current demand arrays, the remaining capacity arrays, and the violated set will update accordingly based on the chosen action. The episode continues until there is no more request in the violated set or the predefined number of maximum steps per episode has been reached.

III. Reinforcement Learning Model

A. Action space

Figure 2 shows the design of the action space. The goal of strategic slot scheduling is to minimize the difference between airlines' requested time slots and the assigned time slots with respect to airports' capacity, resulting in the assigned time slots varying around the requested slots. Therefore, we design the agent's actions to be either shifting the request to a later or earlier slot. A natural intuition is to make the action space to be a space of all 288 time slots and let the agent decide which time slot to allocate. However, this approach has two drawbacks. First, too many actions can significantly hinder the agent's training efficiency [26]. Secondly, it is not reasonable in reality to shift a request to a time slot too far compared to the initially requested time slot, e.g. shifting a request from 6 am to 6 pm. Therefore, it is reasonable to limit the number of actions, i.e. limit the maximum time slot displacement. We design the action space of the agent to move forward by n time slots $(+n)$ or backward by n time slots $(-n)$ or keep the time slot the same compared with the current time slot of the request as per figure 2. Allowing to displace a maximum of n time slots per step does not necessarily mean that a request cannot be displaced further than n time slots. If a request is displaced $k \leq n$ time slots but is still in the violated set, then it may be further displaced in another step.



t - n
...
t - 1
t
t + 1
...
t + n
Move backward by n time slot
...
Move backward by 1 time slot
Keep the time slot the same
Move forward by 1 time slot
...
Move forward by n time slot

Fig. 2. Action space of the Reinforcement Learning agent.

B. State observation

To make decisions, the agent must have information about the current demand and capacity of the time slots. The capacity is assumed to be fixed across all time slots. However, the demand can change depending on the agent's course of action. Since the capacity is fixed, both the capacity and demand information can be encapsulated in the remaining capacity c'_t.

Giving the agent the remaining capacity of all 288 time slots would be intuitive thinking. However, too much irrelevant information may result in inefficient training as the agent cannot identify which piece of information has the highest contribution to the outcomes. For example, the agent does not necessarily need to know the remaining capacity of time slot 200 when the current investigating request is assigned at time slot 100. To prove this claim, we perform training on two modifications of state observation:

- Full State Observation: which is an array of size 578, consisting of the current time slot of the departure/arrival side and the remaining capacity of all 288 time slots at the departure/arrival airport.
- Local State Observation: which is an array of size $2 * (2n + 1)$, consisting of the remaining capacity of the n time slots before the current time slot of the departure/arrival side, the remaining capacity of the current time slot of the departure/arrival side, the remaining capacity of the n time slots after the current time slot of the departure/arrival side.

C. Reward design

The design of the reward affects the training efficiency of the Reinforcement Learning agent [27]. A straightforward reward design for the slot scheduling problem is to penalize the displacement of requests and give a penalty for the total unaccommodated requests as the nature of the problem is to minimize the total schedule displacement and minimize the number of unaccommodated requests. Therefore, we come up with the first two components of the reward:

- $R_{local} = z * (-|t - t_m|)$, where t is the newly assigned time slot for the request and t_m is the original time slot of the request. This is the penalty for displacement that resembles the widely adopted cost function in the slot scheduling problem. The constant number z is to normalize the reward to a smaller number to increase training efficiency. This reward is given to the agent every time steps.
- $R_{global} = -u$ or $R_{global} = v$, where u is the total number of unaccommodated requests at the end of each episode. This is the penalty for failing to accommodate all requests based on the number of unaccommodated requests left. If there is no unaccommodated request at the end of each episode, the agent receives a positive constant reward v. R_{global} is only given to the agent when an episode ends.

To encourage the agent to solve the problem faster and prevent any undesired behavior of prolonging the episode, a third component is added to the reward structure:

- $R_{time\_step} = -p$, where p is a constant number. This reward is given to the agent at every time step.

In the literature, the reward design with the above components is currently adopted in the pre-tactical airport slot reallocation problem [13], [21], [22]. One problem with this reward design is that for every time step, besides the penalty per time step, the agent will only receive 0 if the agent decides to keep the time slot the same or receive a penalty for shifting the request. The further the displacement, the larger the penalty. This will discourage the agent from trying to solve the request and only maintain the original time slot of the request. The agent needs to learn the behavior of clearing a request from the violated set. Therefore, to overcome the problem, we add one more reward component to the agent:

- $R_{solving} = +s$, where s is a constant number. This reward is given to the agent if the agent can clear the current considered request out of the violated set. The reward encourages the agent to learn the behavior of clearing a request out of the violated set. One should note that balancing between the $R_{solving}$ and the R_{local} is important as the agent may learn to create more unaccommodated requests, and prolong the episode to earn more $R_{solving}$. We use two reward designs, Reward Design A (with $R_{solving}$) and Reward Design B (without $R_{solving}$), to validate our claim that giving the $R_{solving}$ is beneficial for training efficiency.

Figure 3 shows the two designs of the state observation, the Full State Observation and the Local State Observation, and the two reward designs, Reward Design A and Reward Design B. We perform experiments with different combinations of state observation and reward design.

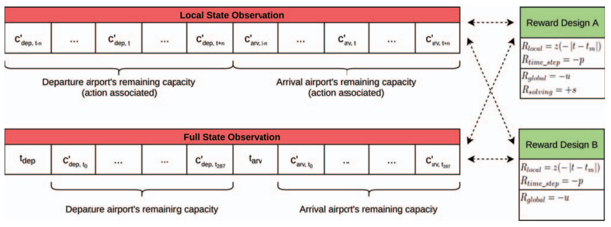

Local State Observation
$c'_{dep,\,t-n}$ … $c'_{dep,\,t}$ … $c'_{dep,\,t+n}$ $c'_{arv,\,t-n}$ … $c'_{arv,\,t}$ … $c'_{arv,\,t+n}$
Departure airport's remaining capacity (action associated)
Arrival airport's remaining capacity (action associated)
Full State Observation
t_{dep} $c'_{dep,\,t_0}$ … … $c'_{dep,\,t_{287}}$ t_{arv} $c'_{arv,\,t_0}$ … … $c'_{arv,\,t_{287}}$
Departure airport's remaining capacity
Arrival airport's remaining capacity
Reward Design A
$R_{local} = z(-|t-t_m|)$
$R_{time\_step} = -p$
$R_{global} = -u$
$R_{solving} = +s$
Reward Design B
$R_{local} = z(-|t-t_m|)$
$R_{time\_step} = -p$
$R_{global} = -u$

Fig. 3. The different combinations of the Full State Observation/Local State Observation and the Reward Design A/Reward Design B.

IV. Experiments

A. Training scenarios and settings

Figure 4 shows the flow diagram from the training phase to the testing phase. The data are generated randomly with 1000 requests, the approximate number of movements per day at Changi airport. We assume two hypothetical distributions of the requested time since we do not have access to the original requests from airlines. We assume two peak demand periods for movements at time slot 72, the time interval 0600-0605, and time slot 216, the time interval 1800-1805, the first distribution is the normal distribution $X \sim \mathcal{N}(\mu = 72, \sigma)$ and the second distribution is the normal distribution $Y \sim \mathcal{N}(\mu = 216, \sigma)$. We consider the standard deviation $\sigma = 60$ and $\sigma = 36$ to create medium and high-density scenarios. The capacity of the airports is assumed to be fixed $c_t^a = 6, \forall a \in A$, $\forall t \in T$.

Settings 1, 3, 4, and 5 are the different combinations of the full state observation/local state observation and the Reward Design A/Reward Design B and are trained with the high-density scenarios. Since only Setting 1 converges, we train an additional Setting 2, which is the exact combination of Setting 1 but the training data are medium-density scenarios. We summarize the 5 training settings as below:

- Setting 1: Reward design A, local state observation, and high-density scenarios in training ($\sigma = 36$).
- Setting 2: Reward design A, local state observation, and medium-density scenarios in training ($\sigma = 60$).
- Setting 3: Reward design B, local state observation, and high-density scenarios in training ($\sigma = 36$).
- Setting 4: Reward design A, full state observation, and high-density scenarios in training ($\sigma = 36$).
- Setting 5: Reward design B, full state observation, and high-density scenarios in training ($\sigma = 36$).

B. Learning algorithms and hyper-parameters

During the training phase, we use the Deep Q-Network (DQN) as the learning algorithm across the 5 settings. The algorithm is well-suited for problems with discrete action spaces. We also make use of two techniques, experience replay and target network, for efficient data utilization and training stability. Readers can refer to [28] for further details of the algorithm.

The hyper-parameters for training DQN are buffer capacity 50000, batch size 32, discount rate 0.99, learning rate 0.0001, target update cycle 10000, and number of training steps 10000000. The number of actions n is 6, which is equivalent to a maximum displacement of 30 minutes per time step. The z in R_{local} is 0.1, the v in R_{global} is 30. The p in R_{time_step} is 0.8 to balance with the average reward gained after solving a request. Balancing the reward components is important to avoid the agent creating more unaccommodated requests to gain higher rewards.

Since the DQN algorithm leads to convergence on Setting 1 and Setting 2, we additionally train the Proximal Policy Optimization (PPO) algorithm with the two settings for performance comparison with DQN. PPO is well-known for tasks with continuous action spaces. However, the continuous action spaces can be binned as discrete actions allowing PPO to be applied to our problem. Readers can refer to [29] for further details of the algorithm.

The hyper-parameters for training PPO are batch size 64, discount rate 0.99, learning rate 0.0003, number of epochs 10, and number of training steps 10000000. The number of actions and the reward components are kept the same with DQN.

We refer to the models DQN (Setting 1), DQN (Setting 2), PPO (Setting 1), and PPO (Setting 2) as Model 1, Model 2, Model 3, and Model 4, respectively.

C. *Convergence and performance metrics*

After the training phase is finished, we evaluate the training efficiency by analyzing the convergence of not only the reward but also the episode length. Gaining a high reward is the goal of the agent but it does not guarantee that the goals of the problem are aligned as the agent can develop behavior such as prolonging the episode by creating more unaccommodated requests to gain more reward. We summarize the two metrics for convergence analysis as below:

- Average Episode Reward $= \frac{\sum \text{Episode Rewards}}{\text{Number of Episodes}}$
- Average Episode Length $= \frac{\sum \text{Episode Lengths}}{\text{Number of Episodes}}$

During the testing phase, the generated data for each run will use either the medium or high-density scenarios to test the agent's ability to generalize on unseen scenarios. We consider the following performance metrics for performance comparison:

- Total schedule delay $M_1 = \sum_{m \in M} |t - t_m|$.
- Maximum displacement across all requests $M_2 = \max\limits_{m \in M} |t - t_m|$.
- Number of unaccommodated requests $M_3 = u$.
- Average displacement per request $M_4 = M_1/U$, where U is the number of unaccommodated requests at the beginning of each episode.
- t-statistic and p-value to measure the difference between the results obtained from Model 1 and Model 2. If Model 1 can achieve comparable results with Model 2 on medium-density scenarios, it means that Model 1 can generalize well on unseen scenarios, and vice versa. If the difference is not significant, we can conclude that the models can generalize to unseen scenarios. The same procedure is performed on Model 3 and Model 4.

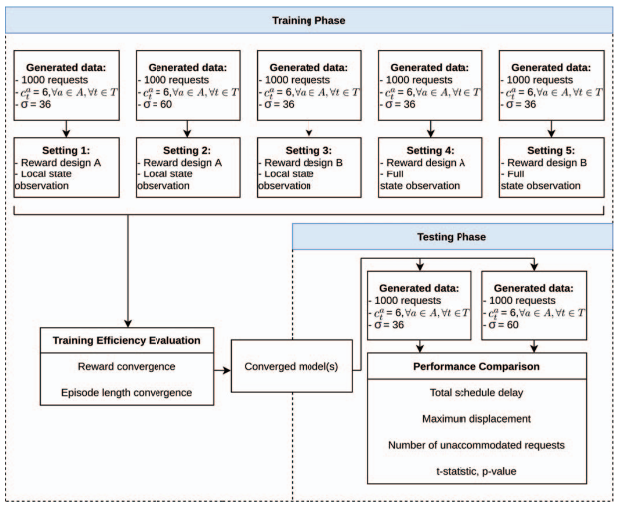


Fig. 4. The training phase and the testing phase flow diagram.

V. Results and Discussion

A. *Training results*

Figure 5 shows the training results of the 5 settings. From the graphs, considering the DQN algorithm, we can see that Setting 1 and Setting 2 achieve convergence stably based on the increasing trend in the average episode reward and the decreasing trend in the average episode length. In Setting 1, the reward obtained after convergence at the end of the training phase is 1.468. This is a reasonable score since with the reward per time step when solving a request $R_{local} + R_{solving} + R_{time\_step} = 0.1(-|t - t_m|) + 1 - 0.8, \in \big[0.1 * (-6) + 1 - 0.8, 0.1 * (-1) + 1 - 0.8\big] = \big[-0.4, 0.1\big]$, the average number of unaccommodated requests per episode of 369.65, and the reward for clearing all requests of 30, the reward can vary between $[-0.4 * 369.65 + 30, 0.1 * 369.65 + 30] = [-117.86, 66.965]$. Similarly for Setting 2, the reward of 36.151 is also reasonable since with the average number of unaccommodated requests per episode of 249.80, the range of the reward is $\big[-0.4 * 249.80 + 30, 0.1 * 249.80 + 30\big] = \big[-69.920, 54.980\big]$. The PPO algorithm achieves similar trends for both settings with slightly lower results, -10.869 for Setting 1 and 27.051 for Setting 2.

For each episode, the average number of unaccommodated requests is 249.80 and 369.65 for medium and high-density scenarios, respectively. Model 1 and Model 3 achieve respectively 399.81 and 412.89 average episode length, which is comparable with the figure of 369.65 for high-density scenarios. Similarly, Model 2 and Model 4 achieve respectively

256.72 and 262.51 average episode length, which is comparable with the figure of 249.80 for medium-density scenarios. This shows that the agent does not need to frequently re-visit the unaccommodated requests.

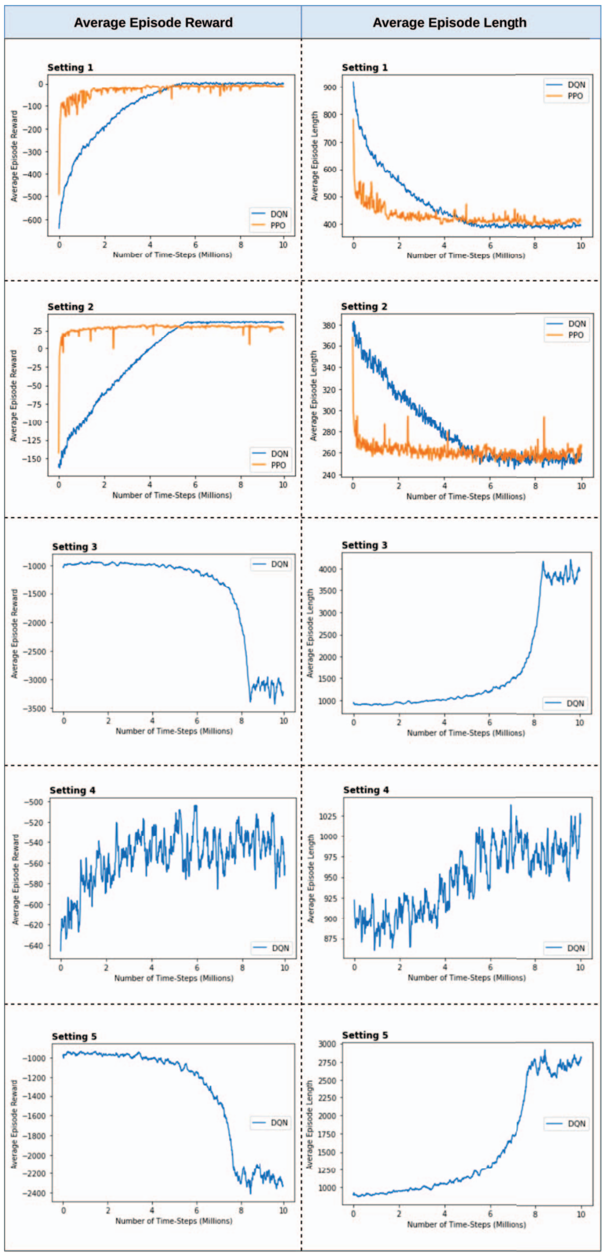


Fig. 5. Training results of the 5 settings.

B. Testing results

Table I and Table II show the testing results of the 4 models on 50 medium-density scenarios and 50 high-density scenarios, respectively. The scenarios are the same across 4

TABLE I
TESTING RESULTS (MEDIUM-DENSITY)

	DQN (Setting 1)	DQN (Setting 2)	PPO (Setting 1)	PPO (Setting 2)
U	246.26 (25.99)			
M_1	355.84 (40.90)	355.18 (39.79)	392.72 (44.00)	391.94 (46.42)
M_1^*	355.99	355.18	392.72	391.94
M_2	4.30 (0.75)	4.36 (0.69)	5.44 (0.75)	6.18 (0.65)
M_3	0.02 (0.14)	0.00 (0.00)	0.00 (0.00)	0.00 (0.00)
M_4	1.45 (0.08)	1.44 (0.07)	1.60 (0.07)	1.59 (0.08)

Each value cell indicates the mean and the standard deviation (in parentheses) of the corresponding metric.

TABLE II
TESTING RESULTS (HIGH-DENSITY)

	DQN (Setting 1)	DQN (Setting 2)	PPO (Setting 1)	PPO (Setting 2)
U	377.32 (33.41)			
M_1	750.82 (81.17)	768.04 (88.85)	785.30 (88.59)	729.58 (76.61)
M_1^*	751.11	769.53	786.60	746.67
M_2	7.28 (1.78)	8.46 (2.53)	9.76 (1.63)	10.22 (2.53)
M_3	0.02 (0.14)	0.08 (0.34)	0.08 (0.39)	0.84 (0.99)
M_4	1.99 (0.10)	2.03 (0.14)	2.08 (0.13)	1.93 (0.11)

Each value cell indicates the mean and the standard deviation (in parentheses) of the corresponding metric.

models. During testing, there are cases in which the models do not solve all the unaccommodated requests, which leads to lower total schedule delay. Although lower total schedule delay is a positive indicator of good performance, achieving this when there are still unaccommodated requests is not encouraged. Therefore, we introduce the adjusted total schedule delay $M_1^* = M_1 + M_3 * (M_2 + 4 * \sigma(M_2))$ to correct the total schedule delay in such occurrences. $M_2 + 4 * \sigma(M_2)$ represents the furthest displacement based on the obtained statistics.

For medium-density scenarios, Model 2 shows the best overall performance with an average total schedule delay of 355.18 without having any unaccommodated requests left. For high-density scenarios, although Model 4 achieves the lowest average total schedule delay/adjusted with 729.58/746.67, this model has a significantly high average number of unaccommodated requests compared to the other three models. As for the remaining three models, Model 1 has the best performance, with an average total schedule delay of 750.82 (751.11 after adjustment), an average maximum displacement of 7.28, an average number of unaccommodated requests of 0.02, and an average displacement per request, 1.99. Overall, DQN models have better performance compared to PPO models.

We compute the t-statistic and p-value on the total schedule delay of the 50 runs of each type of scenario between Model 1 and Model 2. The t-statistics of 0.0810/-1.0016 and the p-values of 0.9356/0.3190 for medium/high-density scenarios suggest that there is no significant difference between the results obtained from the two models, which imply a fair generalization ability of the DQN models. However, for Model 3 and Model 4, the t-statistics of 0.0854/3.4625 and the p-values of 0.9321/0.0008 for medium/high-density scenarios suggest that the two models do not generalize well on high-density scenarios.

VI. Conclusion

We provide a Reinforcement Learning formulation for the strategic airport slot scheduling problem. We perform training for different combinations of the state observations and reward designs. The results suggest that the combination of the local state observation and the positive reward signal leads to a stable convergence. The obtained DQN models can generalize fairly on unseen scenarios and have an overall better performance compared to the PPO models. In the future, we will expand the problem to include more airports and find more informative features to include in the state observation to improve the performance. Real data can be used for both training and testing phases instead of generated data.

Acknowledgment

Anh Nguyen-Duy is supported by the NTU-Vingroup Graduate Scholarship for his doctoral study, under the supervision of Professor Vu Duong. Some experiment attempts were conducted by the group Agent Aeronauts (Project Experience course CS480) from Singapore Management University.

References

[1] European Commission, *EU Transport in Figures: Statistical Pocketbook*. Belgium, 2013.

[2] International Air Transport Association (IATA). (2014) Fact sheet: Single european sky (ses). [Online]. Available: http://www.iata.org/pressroom

[3] Eurocontrol's Central Office for Delay Analysis (CODA), *CODA Digest: Delays to Air Transport in Europe (Annual 2013)*. Brussels, Belgium: Eurocontrol, 2014.

[4] K. G. Zografos, M. A. Madas, and K. N. Androutsopoulos, "Increasing airport capacity utilisation through optimum slot scheduling: review of current developments and identification of future needs," *Journal of Scheduling*, vol. 20, pp. 3–24, 2017.

[5] J. A. Bennell, M. Mesgarpour, and C. N. Potts, "Airport runway scheduling," *4OR-Quart. J. Oper. Res.*, vol. 9, no. 2, pp. 115–138, Jun. 2011.

[6] S. Ikli, C. Mancel, M. Mongeau, X. Olive, and E. Rachelson, "The aircraft runway scheduling problem: A survey," *Comput. Oper. Res.*, vol. 132, Aug. 2021.

[7] U. Benlic, "Heuristic search for allocation of slots at network level," *Transportation Research Part C: Emerging Technologies*, vol. 86, pp. 488–509, 2018.

[8] K. N. Androutsopoulos, E. G. Manousakis, and M. A. Madas, "Modeling and solving a bi-objective airport slot scheduling problem," *European Journal of Operational Research*, vol. 284, no. 1, pp. 135–151, 2020.

[9] N. A. Ribeiro, A. Jacquillat, and A. P. Antunes, "A large-scale neighborhood search approach to airport slot allocation," *Transportation Science*, vol. 53, no. 6, pp. 1772–1797, 2019.

[10] R. Zhang, A. Prokhorchuk, and J. Dauwels, "Deep reinforcement learning for traveling salesman problem with time windows and rejections," in *2020 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 2020, pp. 1–8.

[11] Y. Wang, H. Liu, W. Zheng, Y. Xia, Y. Li, P. Chen, K. Guo, and H. Xie, "Multi-objective workflow scheduling with deep-q-network-based multi-agent reinforcement learning," *IEEE Access*, vol. 7, pp. 39 974–39 982, 2019.

[12] H. Ali, D.-T. Pham, S. Alam, and M. Schultz, "A deep reinforcement learning approach for airport departure metering under spatial–temporal airside interactions," *IEEE Transactions on Intelligent Transportation Systems*, vol. 23, no. 12, pp. 23 933–23 950, 2022.

[13] C. Yutong, H. Minghua, X. Yan, and Y. Lei, "Locally generalised multi-agent reinforcement learning for demand and capacity balancing with customised neural networks," *Chinese Journal of Aeronautics*, vol. 36, no. 4, pp. 338–353, 2023.

[14] Y. Chen, Y. Xu, and M. Hu, "General multi-agent reinforcement learning integrating heuristic-based delay priority strategy for demand and capacity balancing," *Transportation Research Part C: Emerging Technologies*, vol. 153, p. 104218, 2023.

[15] D.-T. Pham, L. L. Chan, S. Alam, and R. Koelle, "Real-time departure slotting in mixed-mode operations using deep reinforcement learning: A case study of zurich airport," 2021.

[16] S. Limin, P. Due-Thinh, and S. Alam, "Multi-agent deep reinforcement learning for mix-mode runway sequencing," in *2022 IEEE 25th International Conference on Intelligent Transportation Systems (ITSC)*. IEEE, 2022, pp. 586–593.

[17] K. Tumer and A. Agogino, "Distributed agent-based air traffic flow management," in *Proceedings of the 6th international joint conference on Autonomous agents and multiagent systems*, 2007, pp. 1–8.

[18] A. M. F. Crespo, L. Weigang, and A. G. de Barros, "Reinforcement learning agents to tactical air traffic flow management," *International Journal of Aviation Management*, vol. 1, no. 3, pp. 145–161, 2012.

[19] C. Spatharis, A. Bastas, T. Kravaris, K. Blekas, G. A. Vouros, and J. M. Cordero, "Hierarchical multiagent reinforcement learning schemes for air traffic management," *Neural Computing and Applications*, pp. 1–13, 2021.

[20] T. Kravaris, C. Spatharis, A. Bastas, G. A. Vouros, K. Blekas, G. Andrienko, N. Andrienko, and J. M. C. Garcia, "Resolving congestions in the air traffic management domain via multiagent reinforcement learning methods," *arXiv preprint arXiv:1912.06860*, 2019.

[21] Y. Chen, Y. Xu, M. Hu, and L. Yang, "Demand and capacity balancing technology based on multi-agent reinforcement learning," in *2021 IEEE/AIAA 40th digital avionics systems conference (DASC)*. IEEE, 2021, pp. 1–9.

[22] Y. Tang and Y. Xu, "Multi-agent deep reinforcement learning for solving large-scale air traffic flow management problem: A time-step sequential decision approach," in *2021 IEEE/AIAA 40th digital avionics systems conference (DASC)*. IEEE, 2021, pp. 1–10.

[23] T. Duong, K. K. Todi, U. Chaudhary, and H.-L. Truong, "Decentralizing air traffic flow management with blockchain-based reinforcement learning," in *2019 IEEE 17th international conference on Industrial informatics (INDIN)*, vol. 1. Ieee, 2019, pp. 1795–1800.

[24] K. G. Zografos, K. N. Androutsopoulos, and M. A. Madas, "Minding the gap: Optimizing airport schedule displacement and acceptability," *Transportation Research Part A: Policy and Practice*, vol. 114, pp. 203–221, 2018.

[25] S. Wang, J. H. Drake, J. Fairbrother, and J. R. Woodward, "A constructive heuristic approach for single airport slot allocation problems," in *2019 IEEE Symposium Series on Computational Intelligence (SSCI)*. IEEE, 2019, pp. 1171–1178.

[26] A. Yehudai, L. Choshen, L. Fox, and O. Abend, "Reinforcement learning with large action spaces for neural machine translation," *arXiv preprint arXiv:2210.03053*, 2022.

[27] A. Dayal, L. R. Cenkeramaddi, and A. Jha, "Reward criteria impact on the performance of reinforcement learning agent for autonomous navigation," *Applied Soft Computing*, vol. 126, p. 109241, 2022.

[28] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski *et al.*, "Human-level control through deep reinforcement learning," *nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[29] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.