DOI: http://doi.org/10.25728/cs.2023.2.1

METHODS FOR SOLVING SOME PROBLEMS OF AIR TRAFFIC PLANNING AND REGULATION. PART II: Application of Deep Reinforcement Learning

E.L. Kulida[1] and V.G. Lebedev[2]

Trapeznikov Institute of Control Sciences, Russian Academy of Sciences, Moscow, Russia

[1]✉ elena-kulida@yandex.ru, [2]✉ lebedev-valentin@yandex.ru

Abstract. Following part I of the survey, this paper considers the problems of improving the safety and efficiency of air traffic flows. The main challenge in conflict detection and resolution by traditional optimization methods is computation time: tens and even hundreds of seconds are required. However, this is not so much for response in real situations. Deep reinforcement learning has recently become widespread due to solving high-dimensional decision problems with nonlinearity in an acceptable time. Research works on the use of deep reinforcement learning in air traffic management have appeared in the last few years. Part II focuses on the application of this promising approach to the following problems: detecting and resolving aircraft conflicts, reducing the complexity of air traffic at the national or continental level (a large-scale problem), and increasing the efficiency of airport runways through the improved planning of aircraft landings.

Keywords: air traffic management, strategic planning of 4D trajectories, aircraft conflict detection and resolution, reinforcement learning.

INTRODUCTION

Due to the growing air traffic flows and overload of major airports, there is an increasing demand for automating the work of air traffic controllers through developing decision support systems and automated air traffic management systems. Part I of the survey [1] was devoted to the problem of minimizing the number of potential conflicts between aircraft.

The paper [2] overviewed current trends in the application of artificial intelligence (AI) to air traffic management based on conference proceedings and publications on the subject in high-rank journals. Despite significant progress in research on AI for air traffic management, it has not yet become "fully functional" for end users. The slow pace of using AI in air traffic management is due to the critical role of this area: lives are at stake here, and safety is the top priority. Currently, safety in air traffic management is achieved through human participation in the control loop. According to the authors cited, safety will evolve by designing human-oriented systems, understandable to the end user and adaptable to their psychological state. This requires moving toward a more user-oriented, eXplainable AI, where the AI system and the end user can understand each other and interact with each other.

Optimization-based approaches are often computationally expensive, which limits their application. Impressive results were obtained in several research works on air traffic management based on deep reinforcement learning; for details, see [3].

In [4], a reinforcement learning-based model was first formulated and an AI agent was presented to mitigate conflicts and minimize aircraft delays when reaching checkpoints. In [5, 6], different levels of environment uncertainty and traffic density were considered and their effect on the performance of the reinforcement learning-based model to resolve aircraft conflicts was investigated.

If the solutions offered by automatic conflict resolution do not match the dispatchers' thinking or preferences, they are unlikely to be accepted. The paper

[7] developed an interactive AI agent based on reinforcement learning with conflict resolution maneuvers used by a human dispatcher. This approach can potentially increase the dispatcher's level of confidence in the solutions proposed by the agent. The hybrid algorithm proposed in [8] uses known geometric methods in the deep reinforcement learning stage to resolve low-altitude airspace conflicts.

These approaches are effective under low air traffic densities, but centralized architectures cannot cope with intensive air traffic flows when the number of conflicting aircraft increases. In most complex systems, distributed decision-making is believed to have higher efficiency than centralized control. A critical challenge for distributed decision-making in air traffic management is the development of a system that provides recommendations to the aircraft for ensuring safe separation and elimination of uncertainty in real time. Several multi-agent approaches were proposed to deal with high air traffic densities. As was demonstrated in [9–11], multiple agents in a decentralized system can access the complete information about all aircraft in a sector using a scalable and efficient method to achieve high throughput under uncertainty. The agents were trained by one neural network with centralized learning, and a decentralized decision-making scheme was adopted. Many of the proposed agents based on reinforcement learning must be trained in an environment with a fixed number of conflicting aircraft. The computational complexity of learning grows rapidly with increasing the number of conflicting aircraft. In [12], image-based deep reinforcement learning was suggested for resolving aircraft conflicts. Image-based deep learning largely solves the scalability problem. The algorithm can process an arbitrary number of aircraft since their states are replaced by their images. The paper [13] presented an autonomous air traffic management model with aircraft collision prevention in free airspace. A graphical neural network approach to resolving conflicts in free airspace was introduced. Representing each aircraft as a graph node, this approach can handle an arbitrary number of aircraft.

Expectedly, deep reinforcement learning will play a significant role in building future air traffic management systems, but more research is needed here. In real-world applications, deep reinforcement learning raises two significant problems.

The first problem is the safety of air traffic management systems. Modern models use deep neural networks as function approximators. Deep neural networks were discovered to suffer vulnerability to adversarial examples [14, 15] (carefully designed quasi-negligible perturbations that mislead the deep neural network when added to its input data). Furthermore, adversarial examples also appear in the real world without any intruder or maliciously chosen noise [16]. Consequently, it is necessary to scrutinize the mechanisms of adversarial attacks, ways to detect errors or undesirable behavior of AI algorithms, and methods to overcome them.

The second problem consists in explainability. The decision-making process of deep reinforcement learning technology is opaque. Due to its insufficient transparency, pilots and air traffic controllers cannot understand the internal mode of operation. The "black box" nature of the model may prevent users from accepting the predicted outcomes, especially when the model makes key decisions.

The paper [17] presented safety-aware deep Q networks. In this model, two separate neural networks jointly investigate security and optimization costs. According to the authors, their work is the first to consider the vulnerability of the model to adversarial attacks; moreover, the model is safe and well-explainable.

The bottleneck in air traffic management systems is the capacity of the runways of major airports. Aircraft maneuvering control operations in the airport area, such as arrival control, landing sequence, and time planning, are performed by air traffic controllers. The paper [18] systematically overviewed past and most recent theoretical studies of the aircraft landing problem for airports with one or more runways, including their comparison.

The remainder of this paper discusses in detail the deep reinforcement learning approach with application to the following problems: detecting and resolving aircraft conflicts, reducing the complexity of air traffic at the national or continental level (a large-scale problem), and increasing the efficiency of airport runways through the improved planning of aircraft landings.

1. CONFLICT DETECTION AND RESOLUTION BASED ON DEEP REINFORCEMENT LEARNING

1.1. A deep reinforcement learning approach to general problems

Reinforcement learning [19] is a method in which an agent interacts with an environment to maximize a long-term reward. It can be treated as a Markov decision process (S, A, T, R, γ) with the following notations: S is the set of environment states; A is the set of agent's actions; T is the probability of transition between states; R is the reward function; finally, γ is the discount rate.

Being in a state $S_t \in S$ at a training step t, the agent generates an action $A_t \in A$ following a policy π: $S \times A \to R$. Then the agent receives the reward R_t and passes to the next state S_{t+1}. The agent's goal is to maximize the total reward by learning the policy $\pi: S \to A$ that determines the required action in each state.

The states are assessed using a function $V(S)$. The state-value function S_t is updated by the formula

$$V(S_t) \leftarrow V(S_t) + \alpha\left(R_t + \gamma V(S_{t+1}) - V(S_t)\right),$$

where α is the learning rate.

The algorithm involves two neural networks: a critic network and an actor network. The former network updates the parameters of the value function w, whereas the latter network updates the parameters of the policy θ; for details, see [20].

Due to the infinite state space, the state-value function is approximated. The approximation is based on the deep learning of the neural network [21]:

$$\hat{V}(S, w) \approx V_n(S),$$

where w denotes the weights of neurons. In one-step temporal difference learning with the approximate function, the update formula is given by

$$w \leftarrow w + \alpha\left(R_t + \gamma \hat{V}(S_{t+1}, w) - \hat{V}(S_t, w)\right)\nabla_w \hat{V}(S_t, w).$$

The action is selected using a policy $\pi(a_s, \theta)$. For the space of continuous actions, the gradient policy [22] based on the gradient descent method is often adopted. The differentiable policy is defined as $\pi(a_s, \theta)$ and the parameters θ are updated at each step as follows:

$$\theta \leftarrow \theta + \alpha \nabla_\theta \ln \pi(A_t | S_t, \theta) V(S_t).$$

1.2. Conflict detection and resolution (CD&R): problem description [23]

Consider N aircraft in a particular air traffic scenario (Fig. 1). Of these, $(N-1)$ ones are in a sector with radius L, and another aircraft flies into the sector. Each aircraft has an initial position and a target position. The goal of each aircraft is to fly from the initial position to the target position in the minimum time without conflicts with other aircraft. A conflict arises when the distance between two aircraft is below the minimum safe distance (usually 5 nautical miles (nm)). At step t, the position and course angle of the aircraft form the vector $\left(x_n(t), y_n(t), \varphi_n(t)\right)$.

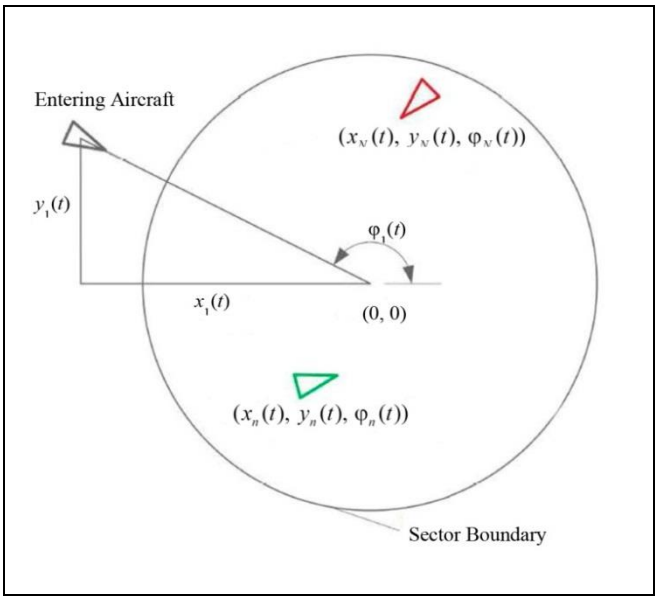

Fig. 1. Conflict detection and resolution problem.

The action for passing to a new state is the new position where the aircraft will fly from its current position.

Figure 2 shows the learning process for one episode.

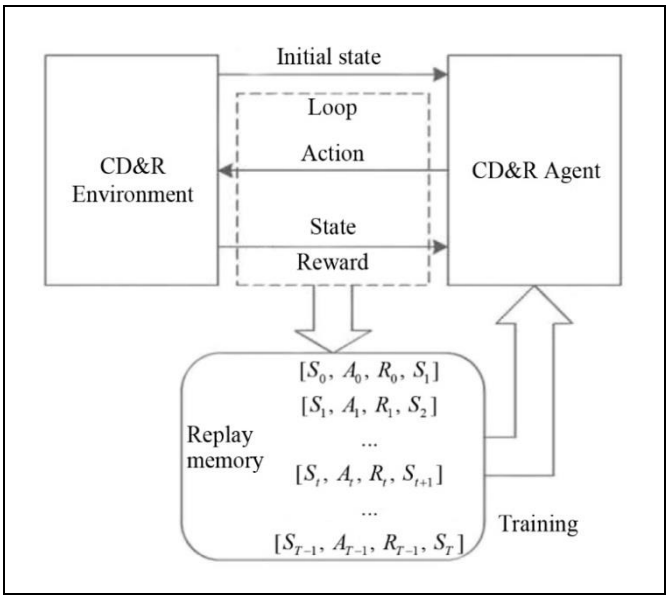

Fig. 2. Training process for one episode.

At the beginning of each training episode, the state S_0 is initialized in the environment. At each step t, the agent obtains the state S_t and implements the action A_t. After that, the environment passes to the state S_{t+1} and returns the reward R_t. This process is repeated until reaching the terminal state S_{t+1}. The value sets

$[S_t, A_t, R_t, S_{t+1}]$ are stored in memory. The agent selects data from memory and learns according to the algorithm.

Possible maneuvers include turns, vertical adjustments, and speed changes. The output state of the environment includes the position of each aircraft in the sector. The control experiment with one aircraft serves for checking that the environment can be used to train agents. Altitude and speed are fixed in this experiment.

The states form a vector of the dimension $N = N_n \times N_p \times N_d$, where N_n is the total number of aircraft (the incoming aircraft plus all aircraft in the sector), N_p is the number of aircraft waypoints (including all waypoints from epy current position to destination), and N_d is the dimension of aircraft location (equals 3). The height dimension is fixed, and the other two dimensions are variable. At each step, the agent perceives the state vector and, after normalization, takes it as the input of neural networks.

The action set is defined as

$$A = \{\rho, \varphi | \rho \in [0, L], \varphi \in [-\pi, \pi]\},$$

where L is the radius of the sector and ρ and φ are the polar radius and angle, respectively. An action is a position described by the two-dimensional polar coordinate. At each step t, the agent chooses an action $A_t \in A$. Depending on this action, the incoming aircraft flies from its current position to A_t.

The agent's goal is to maximize the long-term reward and update the parameters of the neural networks according to the immediate reward. Four rules are used to create the reward function: no conflict between the aircraft, the minimum control time, the minimum course angle change, and the minimum flight distance.

The reward function is given by

$$R_t = \begin{cases} -1 & \text{if a conflict occurs,} \\ |\Delta\varphi_t / \pi| & \text{otherwise,} \end{cases}$$

with the following notations: $\Delta\varphi_t \in [-\pi, \pi]$ is the change of the course angle at step t. Thus, $\left|\frac{\Delta\varphi_t}{\pi}\right| \le 1$, meaning that conflict resolution has the highest priority. Since a change in the course angle will affect the distance, the latter characteristic is omitted in the reward function.

The action is to determine a polar coordinate where the sector's center is the pole and the length shorter than the sector radius is the polar radius. The four outputs of the agent's neural network, $\mu_\rho, \sigma_\rho, \mu_\varphi,$ and σ_φ, are the mean and standard deviation of the polar radius and polar angle, respectively. For learning, the radius and angle are supposed to have the Gaussian distribution: $\rho \sim N(\mu_\rho, \sigma_\rho)$ and $\varphi \sim N(\mu_\varphi, \sigma_\varphi)$. They are generated, and the two-dimensional action is formed. After the agent's neural network is well trained, μ_ρ and μ_φ are taken as ρ and φ, respectively.

The neural networks are trained as follows. For the critic network, the parameter δ is determined to evaluate the chosen action:

$$\delta_t = R_t + \gamma \hat{V}(S_{t+1}, w) - \hat{V}(S_t, w),$$

where R_t is the immediate reward and $\hat{V}(S_t, w)$ and $\hat{V}(S_{t+1}, w)$ are the values of the current and next states, respectively.

The parameters w are updated using the least squares method:

$$w \leftarrow w + \alpha^w \nabla \delta^2.$$

The policy gradient method is applied for the actor network. The policy equation has the form

$$\ln \pi(\rho_t, \varphi_t | S_t, \theta) = \ln \pi(\rho_t | S_t, \theta) + \ln \pi(\varphi_t | S_t, \theta)$$

with the following notations: $\ln \pi(\rho_t, \varphi_t | S_t, \theta)$ is the probability of choosing ρ and φ in the state S_t with parameters θ; $\ln \pi(\rho_t | S_t, \theta)$ is the probability of choosing ρ in the state S_t with the parameters θ; finally, $\ln \pi(\varphi_t | S_t, \theta)$ is the probability of choosing φ in the state S_t with the parameters θ. The parameters θ are updated as follows:

$$\theta \leftarrow \theta + \alpha^\theta \delta_t \nabla \ln \pi(\rho_t, \varphi_t | S_t, \theta).$$

The effectiveness of the proposed approach was demonstrated by numerical simulations. As was declared by the authors, a well-trained agent can generate a solution within 200 ms, whereas previous methods require tens or even hundreds of seconds for calculation. In addition, the turning radius of the aircraft is properly considered, which corresponds to realistic situations.

2. A HYPER-HEURISTIC APPROACH TO REDUCE AIR TRAFFIC COMPLEXITY [24]

The paper [24] considered the problem of improving the airspace structure by air traffic complexity mitigation; see *Section 2.6*.

The hyper-heuristic approach does not simply follow a particular meta-heuristic but also involves flexible integration and adaptive control of low-level heuristics. Several studies confirmed the effectiveness of

Q-learning when selecting an appropriate low-level heuristic at the decision point. *Q*-learning lies in evaluating the best state-action pair using memory with *Q*-tables. Each entry in such a table expresses the long-term value of choosing a particular action in a particular state.

The traditional hyper-heuristic selection structure consists of two levels. The first level contains the problem representation, the state-value function, and a set of low-level heuristics. The second level performs two separate tasks as follows. First, it selects a low-level heuristic and applies it to the solution. Second, it decides to accept or reject the new solution. Selecting appropriate heuristics and decision methods is a non-trivial task when developing a robust hyper-heuristic model. A hyper-heuristic operates without any information needed about the functionality of the low-level heuristics but provides useful feedback on the utilization rate of each heuristic and the change of the objective function. This information is crucial for the learning process.

The algorithm uses a *Q*-learning agent to select a heuristic operator or a low-level heuristic (Fig. 3). The latter is then applied to generate a candidate solution and recalculate its performance in a simulation system.

A decision is then made regarding the candidate solution. If the candidate solution can improve performance, the current solution will be updated. Otherwise, the candidate solution will also be updated with some probability, decreasing with the course of learning. If the candidate solution is not accepted, all changes in the solution space associated with that candidate solution will be undone by a return operation. The *Q*-learning agent determines the reward by checking the current state, the chosen operator, and the evolution of solutions and then updates the *Q*-values in the *Q*-table.

The following heuristic operators are proposed (Fig. 4).

- h_1, a local search that randomly changes the departure time by no more than 5 min (Fig. 4a);
- h_2, a local search that flips the current waypoints in the horizontal plane *XY* along the trajectory (Fig. 4b);
- h_3, a local search that flips the current waypoints in the horizontal plane *XY* perpendicular to the trajectory (Fig. 4c);
- h_4, a local search that randomly deviates the route by changing the position of each waypoint (Fig. 4d);

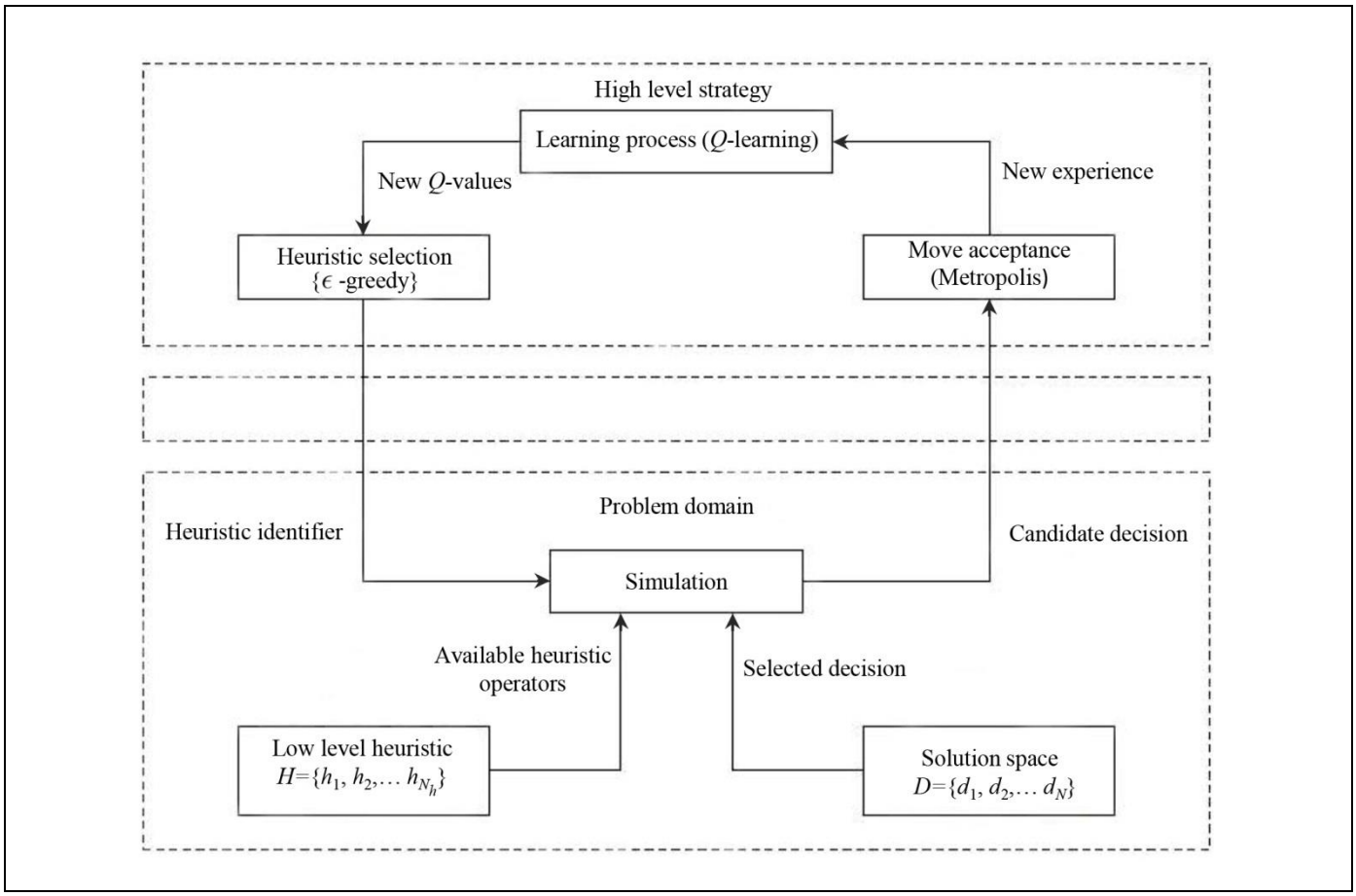

Fig. 3. Framework of Hyper-heuristic based on *Q*-learning with the high-level strategy and problem domain.

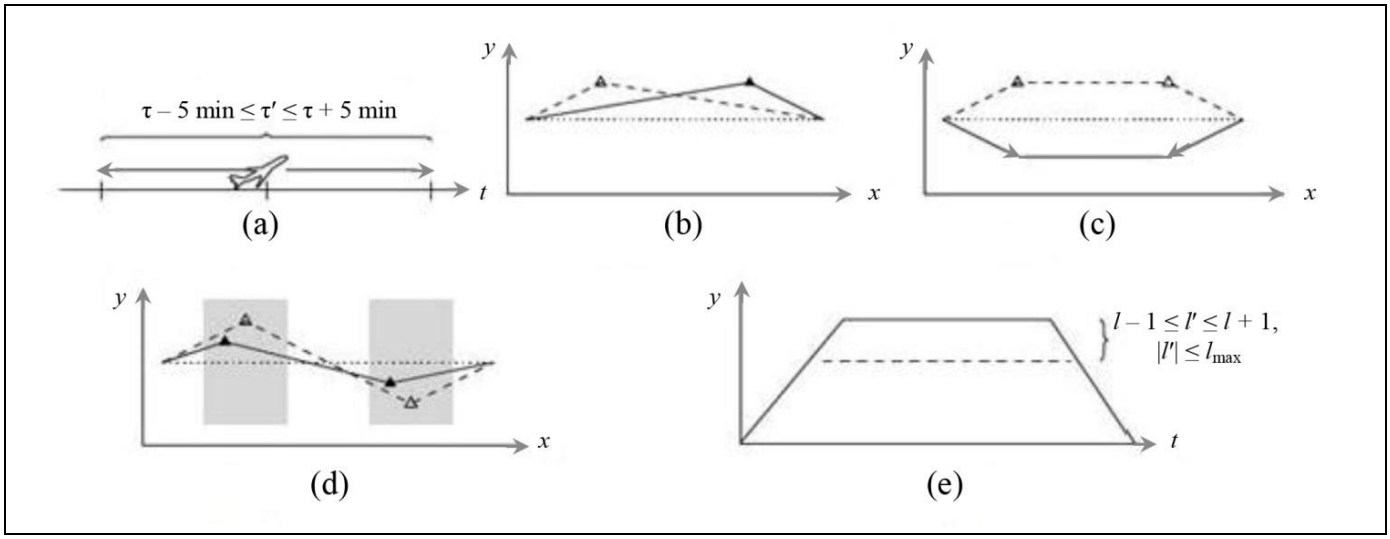

Fig. 4. Representation of intensification heuristic operators which allow the algorithm to refine the search in the vicinity of the current decision:

• h_5, a random selection of a neighbor flight altitude level within the maximum permissible limits (Fig. 4e).

Three generation operators diversify optimization as follows:

• h_6, an operator that randomly changes the departure time within the maximum permissible limits;

• h_7, an operator that randomly adds/removes one or more waypoints according to the problem constraints;

• h_8, an operator that randomly changes the flight altitude level within the maximum permissible limits.

The *Q*-learning agent follows a ϵ-greedy approach to select the heuristic operator based on the *Q*-table. A random action is chosen with probability ϵ and the action based on the *Q*-table with probability $(1-\epsilon)$. At first, ϵ is assigned a maximum user-defined value $\epsilon_{\max}$. During the learning process, it decreases until reaching a minimum user-defined value $\epsilon_{\min}$.

The learning cycle begins by selecting one of the diversification operators h_6, h_7, or h_8. Each state is defined based on the operator applied previously.

Seven states are considered as follows:

• s_0: one of the diversification operators h_6, h_7, or h_8 was previously applied;

• s_1: the operator h_1 was previously applied;

• s_2: the operator h_2 was previously applied;

• s_3: the operator h_3 was previously applied;

• s_4: the operator h_4 was previously applied;

• s_5: the operator h_5 was previously applied;

• s_6: two consecutive operators from the set $\{h_1, h_2, h_3, h_4, h_5\}$ were previously applied without changing the current solution.

The *Q*-table is initialized with *Q*-values according to Table 1.

Some *Q*-values are set to 1 to ensure that a particular heuristic operator can be selected; others are set to 0 to ensure the transition from one state to another.

At each iteration, the system stores the experience containing the current states with the selected operator *h* and the payoff *g* (the difference between the values of the new and previous solutions). At the end of the cycle, the *Q*-learning agent determines the rewards for updating the *Q*-values in the *Q*-table.

At step *t*, the *Q*-values for the found state–action pair are updated as follows:

$$Q(s_t, h_t) = (1-\alpha)Q(s_t, h_t) + \alpha\left(r_t + \gamma \max_{h\in H} Q(s_{t+1}, h)\right),$$

where $\alpha \in [0, 1]$ and $\gamma \in [0, 1]$ are the learning and discount rates, respectively, r_t is the immediate reward, and $H = \{h_i, i = 1,\ldots,8\}$ denotes the set of heuristic operators.

Transfer acceptance determines whether to accept or reject the new solution at each step of the search process. Iterations continue until satisfying a termination criterion.

In [24], the proposed approach was empirically assessed within a real-data experiment for a full day of traffic in the French airspace (8836 trajectories, see Fig. 5).

The initial trajectory plan for the full day of traffic was compared with the final trajectory plan calculated by the proposed algorithm in terms of complexity. The corresponding results are shown in Fig. 6.

Table 1

Q-table initialization

State	Operator							
	h_1	h_2	h_3	h_4	h_5	h_6	h_7	h_8
s_0	1	1	1	1	1	–	–	–
s_1	0	1	1	1	1	–	–	–
s_2	1	0	1	1	1	–	–	–
s_3	1	1	0	1	1	–	–	–
s_4	1	1	1	0	1	–	–	–
s_5	1	1	1	1	0	–	–	–
s_6	–	–	–	–	–	1	1	1

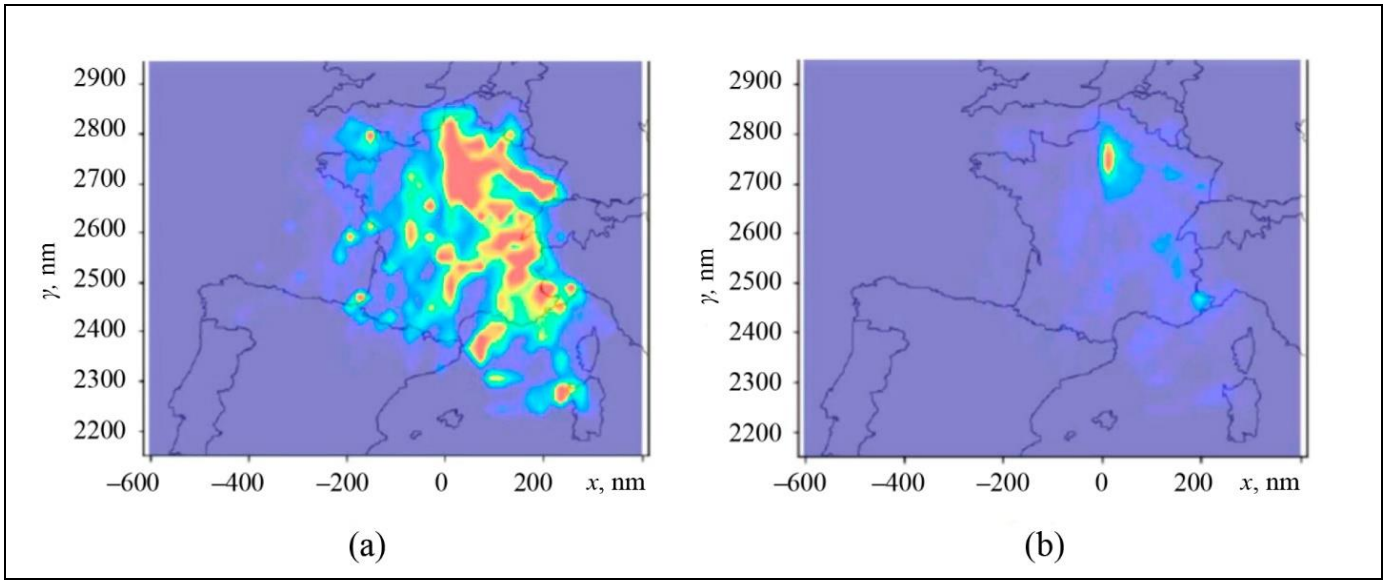

Fig. 5. Complexity map of (a) initial trajectories and (b) final trajectories of a full day of traffic in the French airspace.

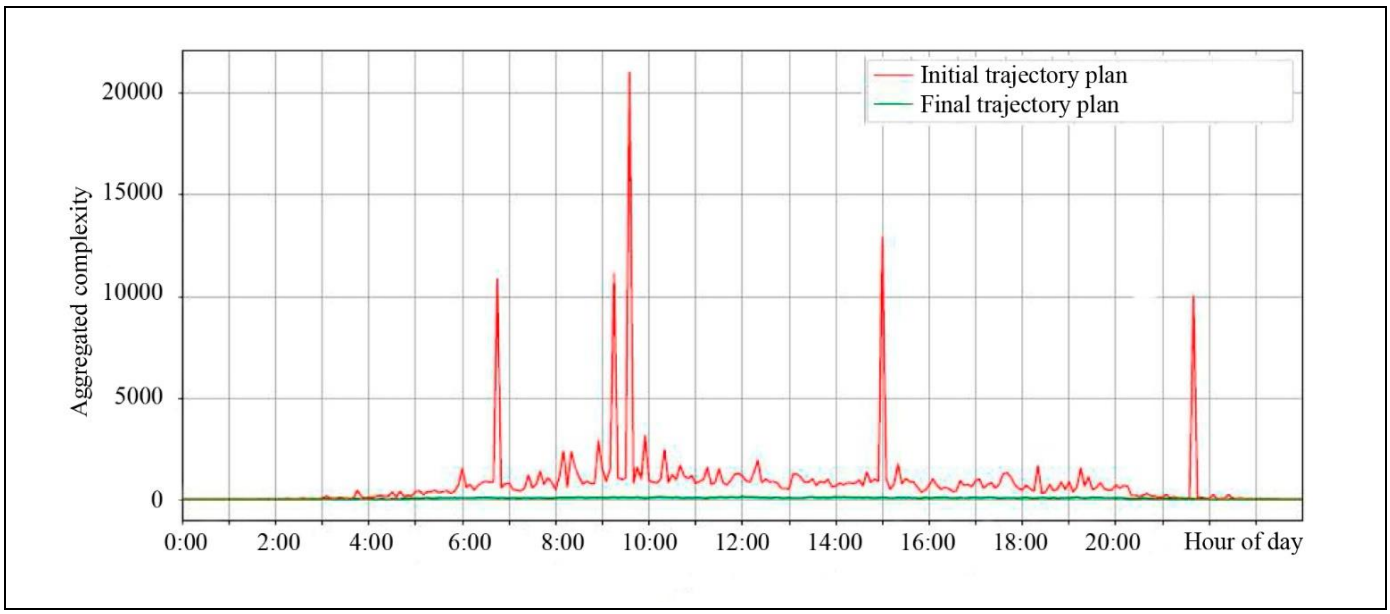

Fig. 6. Comparison of initial complexity and final complexity over time for a full day of traffic in the French airspace.

3. IMPROVING RUNWAY EFFICIENCY USING REINFORCEMENT LEARNING

Airport runways are a major bottleneck in air traffic and a key factor determining airport capacity. Building a new runway is not always possible. One approach to solving the capacity problem is the modernization of the airspace structure and airfield infrastructure. In [25], such an approach was implemented by mathematical modeling. This survey considers another approach: optimizing the use of infrastructure through the improved planning of aircraft landings.

The landing optimization problem is solved in three steps. First, an initial schedule is created on the first-come, first-served basis. Then this schedule is modified during the landing approach phase and finally frozen when the aircraft reaches the final stage of this phase. The initial schedule includes aircraft within the range of the airport landing radar (a time horizon of about 40 min before landing). The update process is executed each time a new aircraft enters the radar range to improve the landing schedule [26].

The most common requirements include a safe separation between consecutive aircraft, allowed time intervals determined by the earliest and latest flight times based on fuel consumption, and priority constraints. Different objective functions serve for increasing runway capacity, meeting schedules, minimizing fuel consumption, etc.

As is known, optimal aircraft sequencing and landing are an *NP*-hard problem [27]. Consequently, the solution time by exact methods grows rapidly with increasing the number of aircraft. Since the first solution [28] published in 1976, several new models and approaches have appeared in the literature, including genetic and heuristic algorithms to obtain a suboptimal but sufficiently efficient solution in an acceptable time [29, 30]. The survey [31] was devoted to some exact approaches to the problem (mainly mixed integer programming), whereas the paper [32] overviewed approximate solution methods, mainly genetic and memetic algorithms. A recent promising approach to the problem is based on reinforcement learning.

The authors [33] considered the problem of planning aircraft takeoffs on a single runway to observe the established time intervals. The problem was modeled as a Markov decision process and solved using the *Q*-learning algorithm [34] as follows. Let the agents be the aircraft and let their states be the aircraft position on the ground depending on its phase (parking, taxiing, and takeoff). The action is to delay the aircraft, and the reward is defined to minimize the delay during taxiing with observing the time intervals established for the aircraft. The algorithm was tested on real data from John F. Kennedy International Airport (JFK, New York), which included departures of 698 flights (two days of operation). Note that 42 training scenarios were generated from the data. According to the results, the algorithm has a performance similar to or greater than that of air traffic controllers.

The paper [35] proposed a framework to model the problem of aircraft sequencing and separation in accordance with the NASA sector-33 application [36]. This air traffic management application contains 35 examples of tasks involving up to 5 aircraft, including speed and route control for aircraft.

The proposed model consists of agents, states, actions, and rewards. There are two types of agents: parent and child. The parent agent's state contains a snapshot of the game screen. The child agent's state contains information about the measurement target, speed and acceleration of the aircraft, and route identifier in addition to information about the N closest agents to allow communication between agents. The actions for the parent/child agent are to change or maintain the route/speed of the aircraft. The reward is designed to penalize conflicting agents (separated by less than 3 nm).

The problem within the model was solved using a hierarchical deep learning algorithm with reinforcement. This algorithm combines the *Q*-learning algorithm [18] and neural networks [5]. It has a hierarchical nature because the actions are executed at two levels: the parent level selects the route and then the child level selects the speed for the aircraft. According to the tests in the NASA application (involving 2–5 aircraft), the proposed approach is viable.

3.1. The Mathematical Model

Let us consider in detail the approach [37]. The distributed algorithm proposed therein is based on *Q*-learning with the parameters optimally tuned by a genetic algorithm. The algorithm was implemented using the sliding window mechanism.

Consider a graph $G(N, L)$, where N and L are the sets of nodes and links, respectively. The node set has two subsets: $N_e \subset N$ (the entry points of the Terminal Maneuvering Areas (TMA)) and $N_r \subset N$ (runways). The final links connecting the runways are also grouped into a link subset: $L_r \subset L$.

Figure 7 shows the network at Paris Charles de Gaulle Airport (CDG). Aircraft enter at LORNI or OKIPA points; there are two runways, 27R and 26L, and two merge points, IF_27R and IF_26L. The two-point merge system is used (IF_27R–RWY_27R and IF_26L–RWY_26L, respectively).

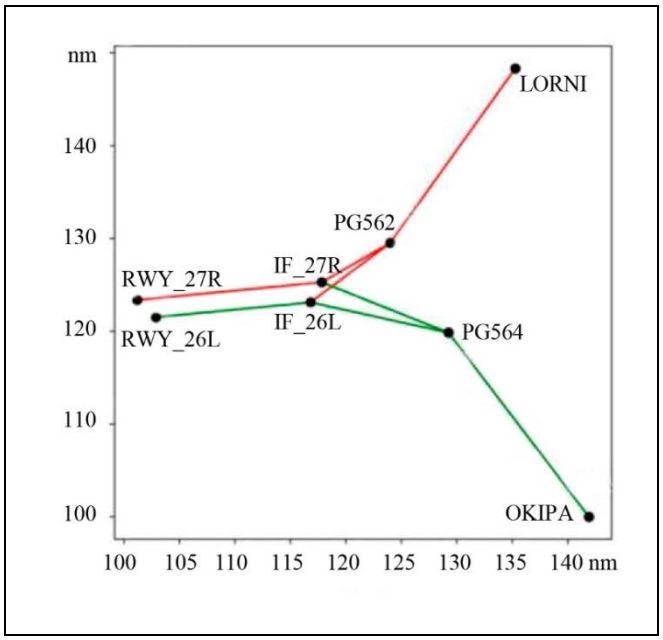

Fig. 7. Simplified STAR model at CDG: aircraft enter at LORNI or OKIPA. Merge points are located on both IF 27R and IF 26L.

For each aircraft, the procedure is executed at a constant speed. This is the landing speed that depends on the wake vortex category.

The point merge system (PMS) structure is presented in Fig. 8. For each aircraft, the length of the PMS arc will be treated as a decision variable for the algorithm.

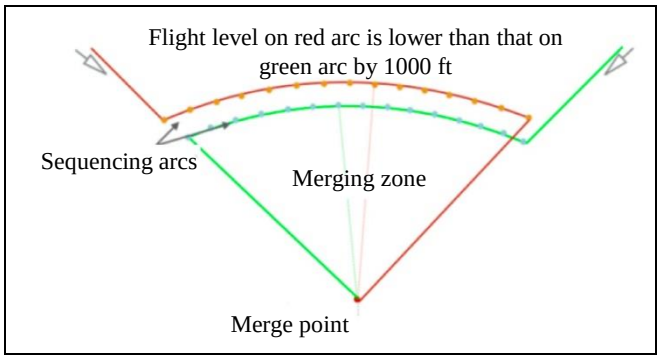

Fig. 8. Merge Point Topology: for each aircraft, the length of the flown sequencing arc is a decision variable.

A flight f is characterized with the following information:

- $V_{0,f}$ is the initial true airspeed of the aircraft;
- $t_{0,f}^{\mathrm{TMA}}$ is the initial entry time in the TMA;
- $r_{0,f} \in N_r$ is the runway on which the aircraft is planned to land;
- t_f^{RTA} is the time at which the aircraft is required to land (the *required time of arrival*, RTA);
- C_f is the wake vortex category.

For each flight f, the following decision variables are considered:

- V_f is the speed of the aircraft;
- t_f^{TMA} is the entry time in the TMA;
- r_f is the runway assigned for landing;
- l_f^{MP} is the length of the merge point arc.

The speed of the aircraft has to stay in a given range of the initial speed:

$$V_f \in V_{0,f} + p\Delta V,$$

where p is the number of increments and ΔV is the speed increment:

$$p \in Z,\ p\Delta V \in \left[\Delta V^{\min}, \Delta V^{\max}\right].$$

Here, $\Delta V^{\max}$ and $\Delta V^{\min}$ are the maximum speed increase and decrease from $V_{0,f}$, respectively, that can be assigned to an aircraft. The minimum speed decrease depends on the wave vortex category.

The entry time decision corresponds to a delay that can be absorbed in the En-Route airspace before the aircraft enters the TMA. In this airspace, the aircraft can be slowed down or accelerated in a given range. As a result, the entry time in the TMA could also change in a given range:

$$t_f^{\mathrm{TMA}} \in t_{0,f}^{\mathrm{TMA}} + p\Delta T,$$

where p is the number of increments and ΔT is the time increment:

$$p \in Z,\ p\Delta T \in \left[\Delta T^{\min}, \Delta T^{\max}\right].$$

Here, $\Delta T^{\max}$ and $\Delta T^{\min}$ are the maximum and minimum time increments from $t_{0,f}^{\mathrm{TMA}}$, respectively, that can be assigned to an aircraft.

To keep a balanced flow between runways, it may sometimes be more appropriate to change the landing runway of an aircraft ($r_f \in N_r$).

As the network contains merge points, one of the decision variables, l_f^{MP}, is the length of the arc that an aircraft will fly in one of the merge points, i.e.,

$$l_f^{\mathrm{MP}} \in p\Delta L,$$

where p is the number of increments and ΔL is the length increment:

$$p \in N,\ p\Delta L \le L_{MP}^{\max},$$

where $L_{\mathrm{MP}}^{\max}$ is the maximum arc length that a merge point can have.

3.2. Description of the Deep Learning Algorithm [37]

This section describes a deep learning algorithm based on the model presented in Section 3.1. It resolves potential aircraft conflicts during heavy traffic in the airport area in a reasonable time.

Each flight is a Markov decision process $MDP\{S, A, P_a, R_a\}$. All decision variables represent the state space *S*. This means that for every aircraft, a state is defined by {speed, entry time in the TMA, PMS arc length, runway assignment}. In each state, the following actions are considered: *A* = {increasing/decreasing the speed, increasing/decreasing the entry time in the TMA, increasing/decreasing the PMS arc length, changing the landing runway, no action}. For states that are not direct neighbors to the current state, the value of the transition function is 0. For neighbor states, the transition function is an equiprobabilistic one:

$$P_a(s, s') = \begin{cases} 0 \text{ if } s' \text{ is not a neighbor of } s, \\ \dfrac{1}{Card(A)} \text{ otherwise,} \end{cases}$$

where $Card(A)$ is the number of elements in *A* (in this case, 8).

Q-learning is a model-free reinforcement learning algorithm. This means that the algorithm does not need a model of the environment, it only interacts with the environment without knowing it. Every aircraft is considered an agent, which makes the algorithm multi-agent.

Q-learning is used to learn the optimal policy of a Markov decision process. This is done by computing the *Q*-function for each aircraft, i.e., representing the expected reward an agent can receive if he takes a given action in a given state. The *Q*-learning used is distributed, meaning that the reward of each agent is treated individually at each iteration.

For each agent, the expected reward $Q(s, a)$ in a given state *s* for a given action *a* is updated as follows:

$$Q(s, a) = Q(s, a) + \alpha\left(R + \gamma \max_{a'} Q(s', a') - Q(s, a)\right).$$

where *s*′ is the new state when the action *a* is taken in the state *s*; *R* is the reward the agent will receive by making the action *a* in *s*; α is the learning rate; finally, γ is the discount factor.

The expected reward $Q(s, a)$ in a given state *s* for a given action *a* is updated at each iteration considering an estimation of the optimal future value $\max_{a'} Q(s', a')$. This is done independently of the policy being followed. Precisely, this is a one-step algorithm since the estimation is done only by looking one iteration ahead.

For a state $s \in S$, an action $a \in A$, and a parameter *T* called temperature, the probability $\pi(s, a)$ to choose *a* in *s* is given by

$$\pi(s, a) = \frac{e^{Q(s,a)/T}}{\sum_{a' \in A} e^{Q(s,a')/T}}.$$

The temperature at iteration *k* is given by a geometric law of the parameter β, i.e., $T_k = T_0\beta^k$, where T_0 is the initial temperature. This temperature sets a trade-off between exploration and exploitation: a relatively high temperature will promote the exploration of the *Q*-table, whereas a low temperature will be in favor of the exploitation of the *Q*-table.

In this distributed *Q*-learning, every aircraft is considered a learning agent and consequently has a *Q*-table. All the *Q*-tables are initialized at a value Q_0 chosen relatively low to enforce the state exploration. This is done on purpose since the reward (and *Q*-table) of an aircraft depends on agents close to it (which can be in conflict). Every agent is seen as an independent learner and does not consider the chosen action of other agents but only their actual states. Therefore, between the two decisions of an agent, its environment may have been changed. An agent can choose the specific action of doing nothing and then its state will not change.

For each aircraft, a reward function is computed and then used by the reinforcement learning algorithm. The reward given at each state and action depends on the other aircraft's state and is computed as the weighted sum of the rewards described below.

All rewards are negative (penalties):

$$R = \omega_{\text{RTA}}\left(R_{\text{RTA}} + 5R_{runway}\right) + \omega_{conflict}(\sum R_{link} + \sum R_{node}).$$

If an aircraft *f* does not land on $r_{0,f}$, its preferred runway, the reward added is 5 times the value R_{runway} weighted by ω_{RTA}. Note that ω_{RTA} and $\omega_{conflict}$ are the algorithm parameters.

Different components of the reward function are described below.

• Required Time of Arrival. All airlines have a schedule for each aircraft and on-time aircraft should have a better reward. Then, a reward corresponding to the absolute difference between the RTA and the real arrival time is added for every aircraft:

$$R_{\text{RTA}} = -\left|t_f^{\text{RTA}} - t_{arrival}\right|.$$

- Runway number

$$R_{runway} = \begin{cases} 0, & \text{landing on the required runway } r_{0,f}, \\ -1, & \text{otherwise.} \end{cases}$$

• Conflicts. The model considers two kinds of conflicts: link conflict, when two aircraft do not respect the wake vortex category separation, and node conflict, when the aircraft do not respect observe the horizontal separation at merge points [38] (3 nm). For each link, at the entrance and the exit, the minimum separation between two aircraft f and g must correspond to Table 2.

Table 2

Minimum separation for link conflict, nm

Category		Leading aircraft, f		
		Heavy	Average	Light
Trailing aircraft, g	Heavy	4	3	3
	Average	5	3	3
	Light	6	5	3

Assuming that $s_{f,g}$ is the minimum separation and $d_{f,g}$ is the actual distance between the leading aircraft f and the trailing aircraft g (Fig. 9), the criticality of a potential conflict, C_{link}, is proportional to the distance between the aircraft. Overtakings are also calculated; if this occurs, then $d_{f,g} < 0$ and the criticality of the conflict is set to –1:

$$C_{link} = \begin{cases} -1 \text{ if } d_{f,g} < 0, \\ -\dfrac{\left| s_{f,g} - d_{f,g} \right|}{s_{f,g}} \text{ if } d_{f,g} < s_{f,g}, \\ 0 \text{ otherwise.} \end{cases}$$

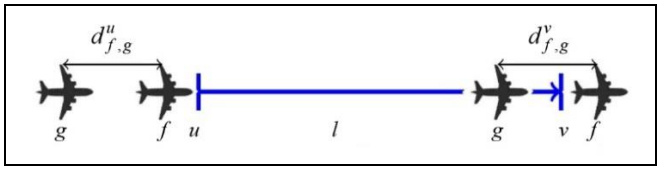

Fig. 9. Link conflict detection based on the comparison of distance between aircraft at the beginning or the end of a link with the separation minima.

The function C_{link} is piecewise linear and continuous, which is necessary for the learning algorithm to know if the conflict is getting better or worse. Since C_{link} can be close to 0, the learning algorithm can improve R_{RTA} instead of resolving the conflict. To prioritize the conflict resolution objective, the value of the reward function for the link is artificially set between –0.3 and –1 using the formula

$$R_{link} = -0.3 + \left(C_{link} \cdot (1 - 0.3) \right).$$

If there is no link conflict between two aircraft f and g, conflicts may still occur on nodes. In TMA, every aircraft has to be separated by 3 nm from others in order to respect the separation distance. As was shown in [38], in many airports, due to the network geometry, the detection area can be reduced to a circle of 2.2-nm radius. As for the links, the criticality of a node conflict is given by

$$C_{node} = \begin{cases} -\dfrac{2.2 - d_{f,g}}{2.2} \text{ if } d_{f,g} < 2.2, \\ 0 \text{ otherwise.} \end{cases}$$

As for the links, the value of the reward function for a node conflict is artificially set between –0.3 and –1 using the formula

$$R_{node} = -0.3 + \left(C_{node} \cdot (1 - 0.3) \right).$$

In this problem, if an aircraft enters the TMA many hours before another one, their decisions can be considered independent. Therefore, the dynamic aircraft landing optimization problem is solved on the basis of a sliding window.

During the optimization process, the aircraft inside the sliding window are divided into four groups:

• completed, the latest landing time is before the starting time of the sliding window;

• ongoing, the earliest entry time is before the starting time of the sliding window (decisions to land have already been made);

• active, the earliest and latest entry times are in the window;

• planned, the latest entry time is after the end of the sliding window.

At each iteration of the sliding window, the optimization algorithm is run on active flights.

Running the algorithm on every active flight in the sliding window is not efficient enough; some of the active flights may have good rewards, and other aircraft may have multiple conflicts. To speed up the optimization process, decisions are changed with a higher priority on aircraft with the worse reward. Those aircraft are indicated as critical flights. They are computed using a threshold that is greater than 70% of the worst aircraft reward. Since these aircraft are learning, their rewards decrease and more and more aircraft become "critical."

The algorithm was successfully tested on data from Paris Charles de Gaulle airport with the total number of aircraft landings artificially increased to 687. A conflict-free solution for a full day of traffic was calculated in less than 30 s, which is acceptable for real-time planning.

CONCLUSIONS

For several decades, extensive research was conducted on decision support automation in ATM systems. Mathematical models developed for this problem either minimize the number of potential conflicts between 4D aircraft trajectories or redistribute aircraft flows to reduce airspace congestion. The number of potential aircraft conflicts is often decreased using one or several methods as follows: shifting flight departure times, regulating airspeeds, changing flight trajectories, and changing flight altitude.

As shown, minimizing the number of potential aircraft conflicts is an *NP*-hard problem. Consequently, various metaheuristic algorithms emerged to solve it. A hybrid metaheuristic approach based on the simulated annealing algorithm, improved by local search methods, was developed for the strategic planning of air traffic flows considering the uncertainty of aircraft positions.

The complexity and scale of minimizing the number of potential conflicts in airspace require new approaches to this problem. Some publications in recent years have been devoted to deep reinforcement learning methods for improving the safety and efficiency of air traffic. The effectiveness of the proposed approaches has been investigated using computational experiments, which have shown encouraging results. Further extensive research is needed to assess the applicability of these approaches in real-world conditions.

REFERENCES

1. Kulida, E.L. and Lebedev, V.G., Methods for Solving Some Problems of Air Traffic Planning and Regulation. Part I: Strategic Planning of 4D Trajectories, *Control Sciences*, 2023, no. 1, pp. 1–12.
2. Degas, A., Islam, M.R., Hurter, C., et al., A Survey on Artificial Intelligence (AI) and eXplainable AI in Air Traffic Management: Current Trends and Development with Future Research Trajectory, *Applied Sciences*, 2022, vol. 12, no. 3, art. no. 1295. DOI: 10.3390/app12031295.
3. Wang, Z., Pan, W., Li, H., et al., Review of Deep Reinforcement Learning Approaches for Conflict Resolution in Air Traffic Control, *Aerospace*, 2022, vol. 9, no. 6, art. no. 294. DOI: 10.3390/aerospace9060294.
4. Brittain, M. and Wei, P., Autonomous Aircraft Sequencing and Separation with Hierarchical Deep Reinforcement Learning, *Proceedings of the 8th International Conference on Research in Air Transportation*, Barcelona, 2018. URL: https://www.reseachgate.net/publication/327287314.
5. Pham, D.T., Tran, N.P., Alam, S., et al., A Machine Learning Approach for Conflict Resolution in Dense Traffic Scenarios with Uncertainties, *Proceedings of the 13th USA/Europe Air Traffic Management Research and Development Seminar (ATM 2019)*, Vienne, 2019.
6. Pham, D.T., Tran, N.P., Alam, S., et. al., Deep Reinforcement Learning based Path Stretch Vector Resolution in Dense Traffic with Uncertainties, *Transportation Research. Part C. Emerging Technologies*, 2021, vol. 135, art. no. 103463. DOI: 10.1016/j.trc.2021.103463.
7. Tran, P.N., Pham, D.T., Goh, S.K., et al., An Interactive Conflict Solver for Learning Air Traffic Conflict Resolutions, *Journal of Aerospace Information Systems*, 2020, vol. 17, no. 6, pp. 271–277.
8. Ribeiro, M., Ellerbroek, J., and Hoekstra, J., Improvement of Conflict Detection and Resolution at High Densities through Reinforcement Learning, *Proceedings of the International Conference on Research in Air Transportation*, Tampa, 2020.
9. Brittain, M. and Wei, P., Autonomous Separation Assurance in a High-Density en Route Sector: A Deep Multi-Agent Reinforcement Learning Approach, *Proceedings of the IEEE Intelligent Transportation Systems Conference (ITSC)*, Aukland, 2019, pp. 3256–3262.
10. Brittain, M., Yang, X., and Wei, P., A Deep Multi-Agent Reinforcement Learning Approach to Autonomous Separation Assurance, *Arxiv:2003.08353v2*, 2020. DOI: https://doi.org/10.48550/arXiv.2003.08353.
11. Brittain, M. and Wei, P., One to Any: Distributed Conflict Resolution with Deep Multi-Agent Reinforcement Learning and Long Short-Term Memory, *Proceedings of the AIAA SciTech 2021 Forum*, Nashville, 2021, p. 1952.
12. Zhao, P. and Liu, Y., Physics Informed Deep Reinforcement Learning for Aircraft Conflict Resolution, *IEEE Transactions on Intelligent Transportation Systems*, 2021, vol. 23, iss. 7, pp. 8288–8301. DOI: 10.1109/TITS.2021.3077572.
13. Mollinga, J., and Hoof, H., An Autonomous Free Airspace En-route Controller Using Deep Reinforcement Learning Techniques, *Arxiv:2007.01599*, 2020. DOI: https://doi.org/10.48550/arXiv.2007.01599.
14. Khan, N.A., Brohi, S.N., and Jhanjhi, N., UAV's Applications, Architecture, Security Issues and Attack Scenarios: A Survey, *Intelligent Computing and Innovation on Data Science*, 2020, vol. 183, pp. 753–760. DOI: 10.1007/978-981-15-3284-9_86.
15. Szegedy, C., Zaremba, W., Sutskever, I., et al., Intriguing Properties of Neural Networks, *Arxiv:1312.6199v3*, 2013. DOI: https://doi.org/10.48550/arXiv.1312.6199.
16. Athalye, A., Engstrom, L., Ilyas, A., and Kwok, K., Synthesizing Robust Adversarial Examples, *Proceedings of the International Conference on Machine Learning*, Stockholm, 2018, pp. 284–293.
17. Wang, L., Yang, H., Lin, Y., et al., Explainable and Safe Reinforcement Learning for Autonomous Air Mobility, *arXiv:2211.13474v1*, 2022. DOI: https://doi.org/10.48550/arXiv.2211.13474.
18. Messaoud, M., A Thorough Review of Aircraft Landing Operation from Practical and Theoretical Standpoints at an Airport Which May Include a Single or Multiple Runways, *Applied Soft Computing*, 2020, vol. 98, no. 12, art. no. 106853. DOI: 10.1016/j.asoc.2020.106853.
19. Sutton, R.S. and Barto, A.G., *Reinforcement Learning: An Introduction*, London: MIT Press, 2017.
20. Degris, T., Pilarski, P.M., and Sutton, R.S., Model-Free Reinforcement Learning with Continuous Action in Practice, *Proceedings of the American Control Conference*, Montréal, 2012, pp. 2177–2182.
21. LeCun, Y., Bengio, Y., and Hinton, G., Deep Learning, *Nature*, 2015, vol. 521, pp. 436–444.

22. Sutton, R.S., McAllester, D.A., Singh, S.P., et al., Policy Gradient Methods for Reinforcement Learning with Function Approximation, *Proceedings of the 12th Conference on Advances in Neural Information Processing Systems (NIPS 1999)*, Denver: MIT Press, 1999, pp. 1057–1063.
23. Wang, Z., Li, H., Wang, J., and Shen, F., Deep Reinforcement Learning Based Conflict Detection and Resolution in Air Traffic Control, *IET Intell. Trans. Syst.*, 2019, vol. 13, pp. 1041–1047.
24. Juntama, P., Delahaye, D., Chaimatanan, S., and Alam, S., Hyperheuristic Approach Based on Reinforcement Learning for Air Traffic Complexity Mitigation, *Journal of Aerospace Information Systems*, 2022, vol. 19, no. 9. DOI: 10.2514/1.i011048.
25. Vishnyakova, L.V. and Popov, A.S., Selection of Airspace Structure and Aerodrome Infrastructure During Their Modernization by Methods of Mathematical Modeling, *J. Comput. Syst. Sci. Int.*, 2021, vol. 60, no. 6, pp. 918–955. https://doi.org/10.1134/S1064230721060174.
26. Bennell, J.A., Mesgarpour, M., and Potts, C.N., Airport Runway Scheduling, *Semantic Scholar*, 2011, vol. 4OR, pp. 115–138. DOI: 10.1007/s10288-011-0172-x.
27. Prakash, R., Piplani, R., and Desai, J., An Optimal Data-Splitting Algorithm for Aircraft Scheduling on a Single Runway to Maximize Throughput, *Transportation Research, Part C: Emerging Technologies*, 2018, vol. 95, pp. 570– 581.
28. Dear, R.G., The Dynamic Scheduling of Aircraft in the Near Terminal Area, *Technical Report no. R76-9*, Flight Transportation Laboratory, Cambridge, MIT, 1976.
29. Kulida, E.L., Genetic Algorithm for Solving the Problem of Optimizing Aircraft Landing Sequence and Times, *Automation and Remote Control*, 2022, vol. 83, no. 3, pp. 426–436.
30. Kulida, E., Egorov, N., and Lebedev, V., Comparison of Two Algorithms for Solving the Problem Aircraft Arrival Sequencing and Scheduling, *Proceedings of the 14th International Conference "Management of Large-Scale System Development" (MLSD)*, September 27–29, 2021. URL: https://ieeexplore.ieee.org/document/9600243.
31. Veresnikov, G.S., Egorov, N.A. Kulida, E.L., and Lebedev, V.G., Methods for Solving of the Aircraft Landing Problem. I. Exact Solution Methods, *Automation and Remote Control*, 2019, vol. 80, pp. 1317–1334.
32. Veresnikov, G.S., Egorov, N.A. Kulida, E.L., and Lebedev,V.G., Methods for Solving of the Aircraft Landing Problem. II. Approximate Solution Methods, *Automation and Remote Control*, 2019, vol. 80, pp. 1502–1518.
33. Soares, I.B., De Hauwere, Y.M., Januarius, K., et al., Departure Management with a Reinforcement Learning Approach: Respecting CFMU Slots, *Proceedings of the IEEE 18th International Conference on Intelligent Transportation Systems*, Las Palmas de Gran Canaria, 2015.
34. Watkins, C.J. and Dayan, P., Q-learning, *Machine Learning*, 1992, vol. 8, pp. 279–292.
35. Brittain, M. and Wei, P., Autonomous Aircraft Sequencing and Separation with Hierarchical Deep Reinforcement Learning, *Proceedings of the International Conference for Research in Air Transportation*, Barcelona, 2018.
36. Colen, J., NASA sector 33 application, 2013. URL: https://www.nasa.gov/centers/ames/Sector33/iOS/index.html.
37. Henry, A., Delahaye, D., and Valenzuela, A., Conflict Resolution with Time Constraints in the Terminal Maneuvering Area Using a Distributed Q-learning Algorithm, *Proceedings of the International Conference on Research in Air Transportation (ICRAT 2022)*, 2022, Tampa, Hal-03701660.
38. Ma, J., Delahaye, D., Sbihi, M., and Mongeau, M., Integrated Optimization of Terminal Manoeuvring Area and Airport, *6th SESAR Innovation Days*, Delft, Netherlands, 2016.

This paper was recommended for publication by A.A. Lazarev, a member of the Editorial Board.

Received November 10, 2022, and revised December 19, 2022. Accepted December 20, 2022.

Author information

Kulida, Elena L'vovna. Cand. Sci. (Eng.), Trapeznikov Institute of Control Sciences, Russian Academy of Sciences, Moscow, Russia
✉ elena-kulida@yandex.ru

Lebedev, Valentin Grigor'evich. Dr. Sci. (Eng.), Trapeznikov Institute of Control Sciences, Russian Academy of Sciences, Moscow, Russia
✉ lebedev-valentin@yandex.ru

Cite this paper

Kulida, E.L. and Lebedev, V.G., Methods for Solving Some Problems of Air Traffic Planning and Regulation. Part II: Application of Deep Reinforcement Learning. *Control Sciences* **2**, 2–14 (2023). http://doi.org/10.25728/cs.2023.2.1

Original Russian Text © Kulida, E.L., Lebedev, V.G., 2023, published in *Problemy Upravleniya*, 2023, no. 2, pp. 3–18.

Translated into English by *Alexander Yu. Mazurov*,
Cand. Sci. (Phys.–Math.),
Trapeznikov Institute of Control Sciences,
Russian Academy of Sciences, Moscow, Russia
✉ alexander.mazurov08@gmail.com