IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received 11 August 2023, accepted 26 September 2023, date of publication 2 October 2023, date of current version 6 October 2023.

Digital Object Identifier 10.1109/ACCESS.2023.3321196

RESEARCH ARTICLE

Application of Improved Q-Learning Algorithm in Dynamic Path Planning for Aircraft at Airports

ZHENG XIANG, (Member, IEEE), HEYANG SUN, (Member, IEEE), AND JIAHAO ZHANG
School of Air Traffic Management, Civil Aviation Flight University of China, Guanghan, Sichuan 618300, China

Corresponding author: Heyang Sun (sunheyang99@outlook.com)

This work was supported in part by the National Nature Science Foundation of China under Grant 62203451.

ABSTRACT Guiding and controlling aircraft within an airport is a decision-making process based on safety and efficiency in a highly dynamic and stochastic environment. Currently, many airports rely on manual monitoring and command to provide appropriate taxiing paths for aircraft. With the increasing complexity of airport structures and flight volumes, there is a need for an algorithm that can autonomously search for the shortest taxiing paths while satisfying the specific taxiing regulations and maintaining safe separations between aircraft in a dynamic scenario. We propose an improved approach based on the Q-Learning algorithm, a reinforcement learning method, to provide taxiing path guidance for aircraft. The Q-Learning algorithm exhibits adaptability in dynamic and stochastic environments. However, the traditional Q-Learning algorithm lacks the iteration stability and computational efficiency required in high-dynamic scenarios, and the shortest paths found often fail to meet the requirements due to the specific regulations of airport control. We first make three improvements to the Q-Learning algorithm to address these challenges. These improvements include optimizing Q-table exploration, resetting initial Q-table values, and introducing a dynamic exploration factor to enhance the algorithm's computational efficiency and accuracy. We also incorporate conflict avoidance strategies related to civil aviation regulations to ensure that the final path adheres to airport control regulations. Finally, we validate the fused and improved algorithm in a gridded airport environment model. Compared to traditional methods, the results demonstrate that the improved algorithm provides more efficient taxiing guidance for aircraft while ensuring operational safety. Furthermore, the algorithm strategically avoids conflicts with other moving aircraft, thereby increasing the utilization of airport taxiing resources.

INDEX TERMS Aircraft navigation, artificial intelligence, reinforcement learning, dynamic path planning, processing time and steps, conflict avoidance.

I. INTRODUCTION

A. MOTIVATION

In recent years, due to the rapid development of civilian aviation, airplanes have become the preferred mode of long-distance travel for people, resulting in a significant increase in air traffic volume. As a result, airports have become larger and more structurally complex. Ensuring safe and efficient taxiing guidance for aircraft has become a vital issue in measuring airport operational efficiency [1], [2].

Clare and Richards [3] proposed a mixed-integer linear programming optimization method to address the coupling problem between aircraft taxiing paths and runway scheduling. Evertse and Visser [4] used mixed-integer linear programming (MILP) to plan taxiing routes for all aircraft on the airport surface, aiming to minimize the taxiing time of aircraft on the ground. Brownlee et al. [5] combined genetic algorithms with the rolling window approach to solving the allocation of taxiing routes for aircraft at the airport. In 2022, Deng et al. [6] proposed a multi-strategy particle swarm and ant colony optimization algorithm, MPSACO, to solve the airport runway planning problem and to avoid conflicts between taxiways and the propagation of conflicts.

The associate editor coordinating the review of this manuscript and approving it for publication was Jiachen Yang.

With the increasing complexity of civil airport structures, traffic density, and specific operational regulations within airports, traditional path-planning algorithms struggle to meet the requirements. Therefore, there is a need for a dynamic path planning method that incorporates airport traffic rules, is accurate, efficient, and capable of handling the heavy workload of airport control [7]. In this paper, we consider the usage rules of taxiways in real-world airports and introduce conflict avoidance strategies for multi-aircraft scenarios. By employing reinforcement learning, we aim to search for the shortest taxiing paths for agents operating in dynamic environments, thereby assisting airport control.

B. RELATED WORK

To enable an agent to perform the specified path planning task in a highly dynamic environment, it is necessary to consider the planning method and the prediction and avoidance of dynamic obstacles during the training process. There have been several notable studies on path planning and conflict prevention. Landry et al. [9] effectively detected and resolved conflicts on airport taxiways by utilizing the attributes of a complex conflict network, and they validated the feasibility of their method through case studies. A distributed multi-agent collision avoidance algorithm based on deep reinforcement learning was proposed by Yu et al. [10], which can calculate and accurately predict the positions and velocities of the surrounding agents with uncertainties. Everett et al. [11] developed a new algorithm based on deep reinforcement learning that allows dynamic agents of various types to learn collision avoidance without assuming any specific behavioral rules. The results showed that the algorithm maintained high performance even with an increasing number of agents. Bailey et al. [12] further subdivided the 2D discretized environment and studied various vertex connectivities to reduce the length of the planned path further. Kiani et al. [13] proposed two heuristic algorithms, Incremental Grey Wolf Optimization (I-GWO) and Extended Grey Wolf Optimization (Ex-GWO), which can rapidly search for optimal collision-free paths between any two points in space.

Many traditional path-planning algorithms, suffer from poor environmental interaction and low generalizability. In contrast, reinforcement learning methods can address these issues by gaining experience through ongoing interaction with the external environment to guide path-planning strategies. As a result, reinforcement learning can accurately accomplish path-planning tasks in complex dynamic environments.

Reinforcement learning is a unique class of machine learning algorithms that have gained significant research focus in artificial intelligence in recent years. This algorithm, developed from fields such as biologically inspired control, psychology, and optimal control, has penetrated various aspects of daily life. In addition to chess and nonlinear control, it has found application in computer vision, natural language processing, autonomous driving, game development, and other areas [14], [15], [16]. Lillicrap et al. [17] introduced the Deep Deterministic Policy Gradient (DDPG) algorithm to address reinforcement learning problems in continuous action spaces, achieving high performance in multiple continuous control tasks. Brittain and Wei [18], [19] formulated the sequencing and separation of air traffic control as a reinforcement learning model and employed a hierarchical deep reinforcement learning algorithm to solve it. They aimed to identify and resolve conflicts among aircraft in complex airspace and evaluated them in the case of the NASA Sector 33 model and BlueSky environments.

Integrating models and planning into reinforcement learning systems allows for a close connection between reinforcement learning and dynamic programming methods, making it an auspicious research direction.

C. PURPOSE AND SIGNIFICANCE

In our research, we have made various improvements to the Q-Learning algorithm in reinforcement learning. After verifying the accuracy and efficiency of the improved algorithm, we considered the real-world aircraft taxiing rules at airports and introduced aircraft conflict avoidance strategies. We combined the improved algorithm to optimize the path planning for departing and arriving aircraft. This fills the research gap where commonly used path search algorithms deviate from civil aviation regulations in determining the shortest taxiing path. The proposed approach assists air traffic controllers in guiding and controlling aircraft movements on the airport surface, reduces their workload, and improves airport operational efficiency, all while ensuring safety.

II. BACKGROUND

A. REINFORCEMENT LEARNING

Like supervised learning and unsupervised learning in machine learning, reinforcement learning (RL) does not refer to a specific model or algorithm but refers to a training method. It is commonly used to solve sequential problems by finding the optimal policy to maximize rewards [20], [21]. The decision-making entity in the reinforcement learning training process is called the Agent, which exists within an environment. Whenever the Agent acts, the environment provides feedback based on that action, and the Agent evaluates the feedback in order to decide on the next action. For reinforcement learning, the foundation of all actions is the reward, and its goal is to maximize long-term, future rewards. Since each agent's action can change the environment, reinforcement learning cannot be trained using a dataset but instead relies on learning from data generated by the natural environment or simulators.

The training process of reinforcement learning can be understood as follows: The Agent interacts with the Environment, receives a State from the environment, and uses a Policy to determine an Action. The Action is then applied to the Environment, and the Environment supplies the following State and Reward obtained as a result of this decision. In a

complex and uncertain Environment, the Agent's goal is to get as much reward from the Environment as possible. The process of Agent-Environment interaction in reinforcement learning is illustrated in the diagram below [22].

1) BASIC ELEMENT

- State: The state of the environment at time t is denoted as St.
- Action: The action taken by the Agent at time t is denoted as At.
- Reward: After the Agent acts, the environment provides immediate feedback to the Agent in the form of rewards, including immediate rewards for the current step and delayed rewards for the long term.

$$R_t = R_{t+1} + R_{t+2} + R_{t+3} + \dots \tag{1}$$

- The discount factor γ is a factor that balances the weight of immediate rewards and delayed rewards. In other words, the update equation for the state value function V_π is the immediate reward plus the value function of the following state multiplied by the reward discount factor. By introducing the discount factor γ, the accumulated return obtained by the Agent is referred to as the discounted return G_t.

$$V_\pi(s) = R_s + \gamma \cdot V_\pi(s) \tag{2}$$

$$\begin{aligned} G_t &= R_{t+1} + \gamma \cdot R_{t+2} + \gamma^2 \cdot R_{t+3} + \dots \\ &= \sum_{k=0}^{\infty} \gamma^k \cdot R_{t+k+1} \end{aligned} \tag{3}$$

- Policy: The mapping from a state to an action is the probability that the Agent will perform a certain action in a particular state of the environment. The state-value function V_π refers to the expected probability of rewards that the Agent obtains when making decisions starting from the current state s according to a given policy π. The action-value function Q_π measures the expected value of the rewards that can be obtained by taking a particular action in the current state of the system.

$$\begin{aligned} V_\pi(s) &= E_\pi [G_t \mid S_t = s] \\ &= E_\pi \left[\sum_{k=0}^{\infty} \gamma^k \cdot R_{t+k+1} \mid S_t = s \right] \end{aligned} \tag{4}$$

$$\begin{aligned} Q_\pi(s, a) &= E_\pi [G_t \mid S_t = s, A_t = a] \\ &= E_\pi \left[\sum_{k=0}^{\infty} \gamma^k \cdot R_{t+k+1} \mid S_t = s, A_t = a \right] \end{aligned} \tag{5}$$

2) MARKOV DECISION PROCESS

The Markov Decision Process (MDP) is a classical mathematical description of sequential decision-making problems [23], [24]. In reinforcement learning, the MDP describes a fully observable environment. The MDP can be represented by a five-tuple $< S, A, P, R, \gamma >$:

- $s \in S$: A set that represents all possible actions in the environment.

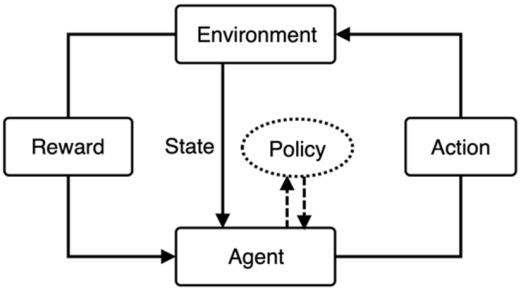

FIGURE 1. Reinforcement learning interaction diagram.

FIGURE 2. Markov decision process.

- a $\in$A: A set representing the actions the Agent can choose from in the environment.
- P : State transition probability, which represents the probability of transitioning from a particular state s_i to state s_{i+1} when taking action a_i.
- R : Represents the immediate reward R(s) in that state.
- γ : The discount factor is a value between 0 and 1 that balances the value of short-term and long-term rewards. A higher discount factor indicates that the agent pays more attention to future rewards, while a lower discount factor prioritizes immediate rewards.

Based on the abovementioned components, MDP provides a formal method for describing sequential decision problems and achieving goals by solving for an optimal policy. An optimal policy refers to selecting the best action at each state to maximize cumulative rewards or minimize cumulative costs. Standard methods for solving MDP problems include value iteration, policy iteration, and reinforcement learning algorithms such as Q-Learning and Deep Q-Network [25], [26]. Through the framework of MDP, reinforcement learning can make decisions in partially observable environments by leveraging the interaction between states, actions, and rewards to learn the optimal policy and obtain maximum long-term returns. This makes reinforcement learning applicable in various domains, such as autonomous driving, robot control, game intelligence.

B. Q-LEARNING

The Q-Learning algorithm [27], [28] is an essential value-based learning algorithm in reinforcement learning. In this algorithm, Q is the expected return when action a is taken at time t, and the environment provides corresponding rewards based on the agent's actions. The main idea of the algorithm is to construct a two-dimensional Q-Table that stores the Q-values for each state (s $\in$ S) and each action (a $\in$A) based on the underlying state-action pairs. The algorithm learns to make better decisions by continuously interacting with the environment, observing the rewards, and updating

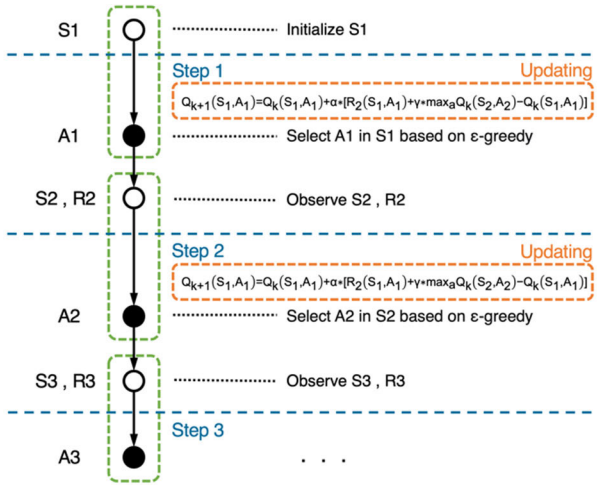

FIGURE 3. Q-Learning algorithm iteration flowchart.

the Q-values in the Q-Table as a function of the action value function.

The Q-Learning algorithm primarily updates the Q-values based on the following formula [29]:

$$Q(s, a) \leftarrow Q(s, a) + \alpha \cdot [R(s, a) + \gamma \cdot \max_a Q(s, a) - Q(s, a)] \quad (6)$$

where Q(s, a) represents the estimated value of the current action-state pair, Q(s′, a) is the estimated value of the next state-action pair. R(s, a) denotes the immediate reward obtained when taking action a in the current state s. max Q(s′, a) selects the action from the next state-action set A that maximizes the Q-value. The learning rate α controls the weight of updates to the Q-values, while the discount factor γ determines the size of future rewards. The ε-greedy strategy balances exploration and exploitation, allowing for exploring new actions while leveraging existing experience. By iteratively updating the Q-table, the algorithm eventually converges to the optimal Q-value function, allowing the optimal policy to be learned.

C. BELLMAN EQUATION

The Bellman equation is a fundamental equation for value functions in reinforcement learning, which describes the relation between a state's value function and the value functions of its neighboring states [30]. With an iterative value function update, the algorithm can gradually converge to the optimal value function, thus learning the optimal decision policy. Specifically, for a state s and an action a, the Bellman equation defines the relationship between the state value function V(s) or the action value function Q(s, a) and the value function of its next state.

Bellman's equation for the state-value function is formulated as follows:

$$V(s) = E[R(s, a) + \gamma \cdot V(s')|s, a] \quad (7)$$

In the context of the equation, V(s)represents the value function of state s, R(s, a) represents the immediate reward obtained by taking action a in state s, γ is the discount factor, V(s′) represents the value function of the following state s′, and E denotes the expectation.

Bellman's equation for the action-value function can be expressed as:

$$Q(s, a) \leftarrow E[R(s, a) + \gamma \cdot \max[Q(s', a')]|s, a] \quad (8)$$

In the given equation, Q(s, a) represents the value function for taking action a in state s, R(s, a) denotes the value function for taking action a in state s, R(s, a) representing the immediate reward gained when taking action a in state s.

One of the critical ideas of the Bellman equation is to relate the value of the value function of the state or the action value function to the value of the successor states. The optimal value function can be progressively approached by iteratively updating the values. In reinforcement learning algorithms, the Bellman equation is a core principle used in value iteration, policy iteration, and other algorithms to help the agent learn the optimal policy.

III. PROBLEM FORMULATION

A. PROBLEM STATEMENT

During the surface operations process at airports, all aircraft operate under the command of air traffic controllers to ensure operations' safety and effectiveness. Our proposed reinforcement learning-based path-finding algorithm aims to find an optimal taxiing path for aircraft on the airport surface given safety by avoiding other aircraft or vehicles in operation. In this work, we selected Shanghai Hongqiao Airport in China as the simulation target. We built an experimental environment, including two closely spaced parallel runways, multiple taxiways and apron areas currently used at Shanghai Hongqiao Airport. We designed five sets of surface operation experiments based on the latest operational regulations, including three experiments involving taxiway conflicts for inbound and outbound aircraft. The simulation experiments aim to predict and avoid conflicts with other aircraft or vehicles in operation and to plan the optimal taxiing path for arriving and departing aircraft. Ensuring safety, the algorithm assists air traffic controllers in commanding aircraft, improving airport operational efficiency.

Traditional path search algorithms such as Dijkstra, A star, and RRT [31], [32], [33] can only find the theoretically shortest route through traversal exploration and formula calculations, and they are difficult to adapt to continuously changing environments. Due to the complex nature of airport surface operations and the existence of special taxi regulations, traditional path search algorithms are not well-suited for aircraft path searches at the airport. In contrast, reinforcement learning methods can incorporate particular search and conflict resolution strategies. The Q-Learning algorithm in reinforcement learning receives real-time feedback from the

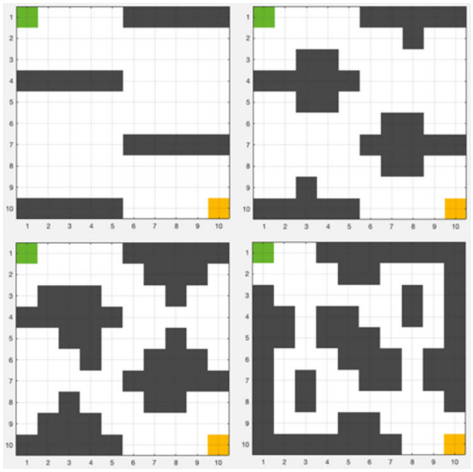

FIGURE 4. Algorithm testing environment model.

environment in each iteration, making it more adaptable to dynamic environments. This paper introduces conflict resolution strategies that integrate airport control regulations into the Agent's movement process, making the simulation results more realistic and closer to actual airport operations. We have also made a series of enhancements to the Q-Learning algorithm to improve the efficiency of the algorithm's search process and the stability of the search after convergence.

B. GRID ENVIRONMENT

The grid-based method is a commonly used path-planning approach for computing paths in a discretized environment. The grid-based method involves partitioning the environment into multiple grid cells, each representing a discrete state space.

For this purpose, we constructed four sets of 10·10 grid maps with obstacle ratios of 20%, 30%, 40%, and 50% to test the performance improvement of the enhanced algorithm. As shown in figure 4, the white area represents the search space available for the agent, and the black area represents the region occupied by obstacles. The four environment models have the same initial position (1, 1) and target position (10, 10) settings.

C. SIMULATION MODEL SETTINGS

1) VARIABLE DEFINITIONS AND OBJECTIVE FUNCTION

For the simulation experiments in this paper, the position state of the agent is determined using Cartesian coordinates. The horizontal axis of the grid is shown as the X-coordinate in the grid map, where values increase from left to right, and the vertical axis of the grid is represented by the y-coordinate, with values increasing from the bottom to the top. The agent searches for a path between the green initial position at (1,1) and the yellow target position at (10,10) on the grid while avoiding black obstacles, and the agent's position state must not exceed the grid boundaries. The goal of path planning is to minimize the search time and the length of the path.

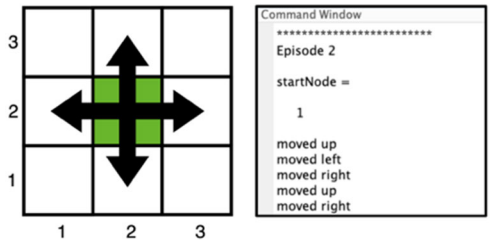

FIGURE 5. Agent search strategy.

We define the convergence state as when the consecutive difference in algorithm steps is not more significant 20. Let T_c denote the time at which the algorithm first converges, and S_c denotes the average step count of the agent's path search after convergence. Therefore, for the agent's path search problem, the objective function can be expressed as:

$$\text{Min}\ \{T_c \& S_c\} \tag{9}$$

2) CONSTRAINTS OF THE PATH PLANNING SIMULATION MODEL

During the operation, we set the search directions of the Agent to four: up, down, left, and right. The search scope covers the maximum values of the grid's horizontal and vertical axes.

The constraint conditions for the search positions of the Agent can be expressed as:

$$P_a = (x \pm 1, y \pm 1) \in (X, Y) \tag{10}$$

For each state of the Agent, the optimal action can be generated using the formula:

$$a^*(s_t) = \max_a Q(s_{t+1}, a) \tag{11}$$

The Q-values in the algorithm are updated using the Bellman equation, and the maximum value in the Q-Table is chosen for the following computation per iteration.

$$\begin{aligned} Q &= Q_{max}(s, a) \\ &= E[R(s, a) + \gamma \cdot \max[Q(s', a')]|s, a] \end{aligned} \tag{12}$$

3) PARAMETERS SETTINGS

For the simulation experiment, the baseline parameter settings are as follows: the initial values in the Q-Table are all set to 0, the maximum episode count is set to 150, the exploration rate ε is 0.1, the learning rate α is 0.1, the discount factor γ is 0.9, and the maximum step length is set to 200. The action reward values are set as follows:

$$\text{Reward} = \begin{cases} 100 & \text{Target position} \\ -10 & \text{Obstacle position} \\ -1 & \text{Other position} \end{cases} \tag{13}$$

The algorithm in this paper was tested and simulated on the Windows 11 operating system with 8GB of RAM and an Intel Core i5-10500 CPU running at 3.10GHz. The simulation software used was MATLAB R2021A.

TABLE 1. Algorithm improvement experimental data: Q-Table exploration guidance.

Environment	Q-Learning Algorithm	Time(s)	Step
Grid-1	QL	106	20
	QL-EG	93	20
Grid-2	QL	112	20
	QL-EG	103	20

IV. SOLUTION APPROACH

A. SIMULATION MODEL SETTINGS

Due to the exploration rate in traditional Q-Learning algorithms, the iteration data becomes unstable after convergence. Therefore, we improved the Q-Learning algorithm in the following three aspects and validated the enhanced algorithm's performance in different complexity environments using multiple environment models.

1) OPTIMIZING Q-TABLE EXPLORATION GUIDANCE

Due to the existence of the exploration rate α, the algorithm always has a certain probability of random exploration, which may cause the agent to choose incorrect actions multiple times, thus affecting the computational efficiency and accuracy. We introduced an exploration-guided strategy for the Q-Table in the algorithm to avoid this situation. The specific method is as follows: when the agent encounters obstacles multiple times at the same position or when the search fails, the corresponding Q-value in the Q-Table for that position should be much smaller than the Q-values for other positions. Before taking a random action, we add a condition check [34] to subtract an enormous value from the Q-Table if an obstacle is encountered during the current search. This helps to eliminate actions prone to encountering obstacles, thus accelerating the convergence speed of the algorithm.

Table 1 shows the performance of the original Q-Learning algorithm (QL) and the algorithm with optimized exploration guidance (QL-EG) in Grid-1 and Grid-2. The improved algorithm demonstrates improvements in terms of time and convergence speed compared to the original Q-Learning algorithm.

2) RESETTING INITIAL VALUES OF Q-TABLE

The traditional Q-Learning algorithm typically initializes all Q-table values to 0. To encourage the agent to select the shortest path, however, we can initialize the Q-values as the reciprocal distance between the current and goal positions. The closer the agent is to the target, the higher the Q-value. This initialization helps the agent to move towards the target location during training initially.

Assuming the coordinates of two points in the coordinate system are (x_1, y_1) and (x_2, y_2), the Euclidean distance d_o and Manhattan distance d_m between the two points can be expressed using formulas:

$$d_o = [(x_1 - x_2)^2 + (y_1 - y_2)^2]^{1/2} \tag{14}$$

$$d_m = |x_1 - x_2| + |y_1 - y_2| \tag{15}$$

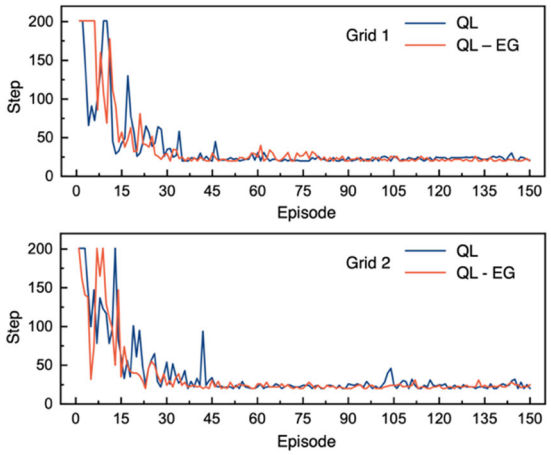

FIGURE 6. Algorithm improvement iteration steps graph: Q-Table exploration guidance.

TABLE 2. Algorithm improvement experimental data: resetting initial values of Q-Table.

Environment	Q-Learning Algorithm	Time(s)	Step
Grid-1	Q=0	106	20
	$Q=1/d_o$	103	20
	$Q=1/d_m$	95	20
Grid-2	Q=0	99	20
	$Q=1/d_o$	98	20
	$Q=1/d_m$	97	20

To validate the efficacy of the enhancements, we performed two sets of experiments in Grid-1 and Grid-2. For each set, we initialized the Q-values in the Q-table using the reciprocal of the Manhattan distance and the reciprocal of the Euclidean distance between the start point and the endpoint.

Table 2 demonstrates the performance of the Q-Table initialization using values of 0, 1/do, and 1/dm in two different environments. While achieving the same path, the improved algorithm outperforms the original algorithm in terms of time and shows improved iteration stability.

3) DYNAMIC EXPLORATION FACTOR

The introduction of the exploration rate ε increases the model's exploration of the environment, improving the likelihood of obtaining the optimal solution. When the algorithm is performing computations per iteration, it will continue to choose the action with the largest value per iteration with probability ε, and randomly choose an action with

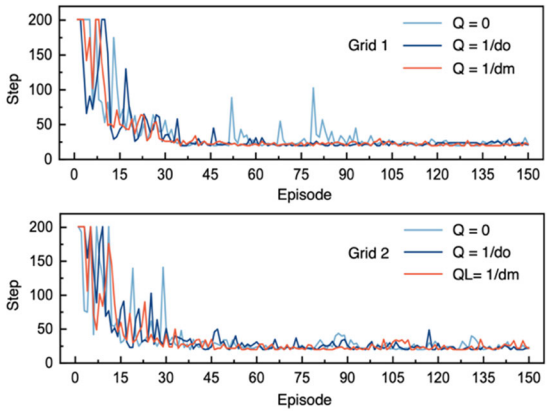

FIGURE 7. Algorithm improvement iteration steps graph: resetting initial values of Q-Table.

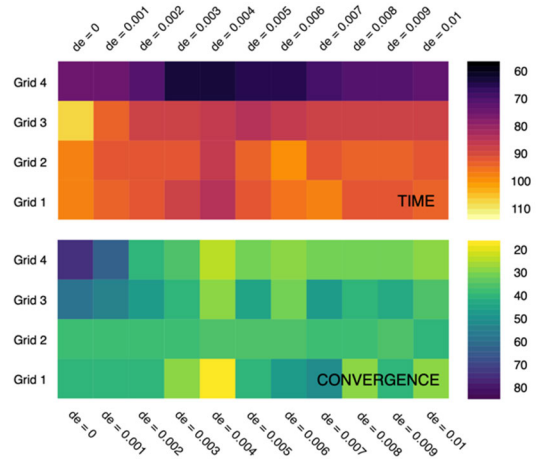

FIGURE 8. Visualization of dynamic exploration factor correlation.

probability 1-ε. The mathematical expression is as follows:

$$A = \begin{cases} \text{value } \max_a Q(a) & P = \varepsilon \\ \text{random action} & P = 1 - \varepsilon \end{cases} \tag{16}$$

In reinforcement learning, the model continuously selects actions with the highest value to execute and update the values. However, some good actions that have yet to be executed in the later stages of exploration may be missed due to random exploration probability. Therefore, regarding the exploration-exploitation trade-off in the Q-Learning algorithm, we propose introducing a dynamic search factor that can be adjusted based on the environment's feedback. If a search succeeds, the search rate decreases by some value to increase the probability of choosing the optimal action in the next exploration by the agent. Conversely, when a search fails, the search rate increases by a specific value to increase the probability of random exploration in the next iteration. In our experiments, we set the maximum episode to 150, exploration rate (ε) to 0.1, and dynamic search factor (dynamic-ε) to 0-0.01. We tested the algorithm's performance with different parameters in the Grid-1 environment and selected the parameters with the best results to apply to the airport simulation environment.

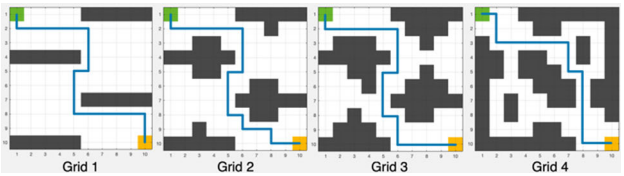

FIGURE 9. Paths searched by improved algorithm in different testing environments.

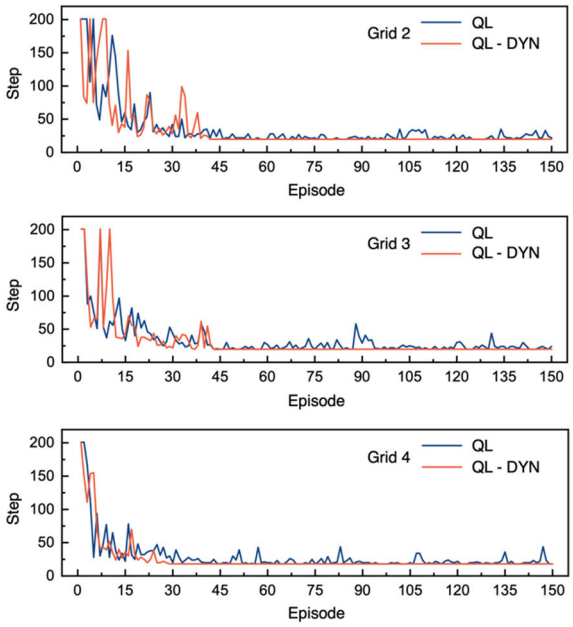

FIGURE 10. Iteration steps graph for algorithm improvement: dynamic search factor.

The impact of different values of dynamic-ε on the algorithm's performance is shown in Figure 9. In the upper graph, darker colors indicate shorter computation time, while in the lower graph, lighter colors indicate faster convergence. It can be observed that when dynamic-ε is set to 0.004, the algorithm performs better in terms of computation time and convergence speed.

B. ALGORITHM COMPARATIVE ANALYSIS

1) COMPARED WITH Q-LEARNING ALGORITHMS

We applied the improved Q-Learning algorithm (QL-DYN) before and after introducing the dynamic search factor in Grid-2, Grid-3, and Grid-4 to validate its applicability further.

From Figure 10, it can be observed that the Agent with the introduction of dynamic-ε achieves faster search speed while maintaining the search path's accuracy and exhibits improved stability after convergence.

2) COMPARED WITH OTHER TRADITIONAL PATH PLANNING ALGORITHMS

After comparing with the Q-Learning algorithm, we compared our enhanced reinforcement learning algorithm (QL-DYN) and traditional path planning algorithms. The

TABLE 3. Experimental data for algorithm improvement: dynamic search factor.

Environment	Q-Learning Algorithm	Time(s)	Step
Grid-2	QL	97	20
	QL − DYN	87	20
Grid-3	QL	113	20
	QL − DYN	88	20
Grid-4	QL	81	18
	QL − DYN	69	18

traditional algorithms used for comparison include Artificial Potential Fields (APF), Ant Colony Optimization (ACO), A* algorithm, and Dijkstra's algorithm.

Ant Colony Optimization (ACO) was proposed by Dorigo in 1996 [35]. It is a stochastic search algorithm that simulates the foraging process of real ants in the natural world. The solving process can be roughly divided into state transitions (path construction) and pheromone updates. The following formula can represent the probability of state transition.

$$p_k(i,j) = \begin{cases} \dfrac{[\tau(i,j)]^{\alpha}\cdot[\eta(i,j)]^{\beta}}{\sum_{\mu\in J_k(i)}[\tau(i,u)]^{\alpha}\cdot[\eta(i,u)]^{\beta}} & j \in J_k(i) \\ 0 & \text{otherwise} \end{cases} \tag{17}$$

After each iteration, the pheromone is updated.

$$\tau(i,j) = (1-\rho)\cdot\tau(i,j) + \sum_{k=1}^{m}\Delta\tau_k(i,j) \tag{18}$$

$$\Delta\tau_k(i,j) = \begin{cases} (C_k)^{-1} & (i,j)\in R^k \\ 0 & \text{otherwise} \end{cases} \tag{19}$$

$$\eta(i,j) = \frac{1}{d_{ij}} \tag{20}$$

In the above formulas, τ represents pheromone, η stands for heuristic information value, α is the weight of pheromone, β is the weight of heuristic information value, ρ is the pheromone evaporation rate, $\tau(i, j)$ denotes the amount of pheromone on the path, with values in the range of [0, 1], m is the number of ants, d_{ij} is the distance between node i and node j, and CK represents the path length searched by ant k in this iteration.

Computer scientist Edsger W. Dijkstra proposed the Dijkstra algorithm in 1959 [36], and it has been widely used in computer science and engineering. Its core principle is to start from a specified node (the source node) and find the shortest paths between it and all other nodes in the graph. During the algorithm's execution, the node with the most minor cost is selected from the priority queue as the next one to be traversed. This process continues until the destination is reached. The essence of the algorithm is breadth-first search, which is a divergent search method, resulting in relatively high space and time complexity.

The A* algorithm combines the Dijkstra algorithm (breadth-first) and the greedy algorithm (depth-first) [37]. While traversing nodes, it records the cost to reach each node from the source and calculates the estimated cost from the current node to the goal node, resulting in better performance and accuracy. Due to the guidance provided by the heuristic function, the A* algorithm often exhibits superior performance. The essential heuristic function for the A* algorithm is shown below:

$$f(n) = g(n) + h(n) \tag{21}$$

In the formula, f(n) represents the overall priority of node n, where g(n) is the cost from the starting point to node n, and h(n) is the estimated cost from node n to the destination.

The Artificial Potential Field (APF) is a virtual force method proposed by Khatib in 1985 [38]. Its fundamental concept is to design the movement of an agent in its surrounding environment as a motion within an abstract artificial gravitational field. The goal point generates "attraction" in the agent, while obstacles generate "repulsion." Ultimately, the agent's movement is controlled by calculating the resultant force. The expressions for traditional gravitational and repulsive fields are as follows:

$$U_{att}(q) = \frac{1}{2}kp_G^2(q) \tag{22}$$

$$U_{rep}(q) = \begin{cases} \dfrac{1}{2}\eta\left(\dfrac{1}{p(q)} - \dfrac{1}{p_0}\right)^2 & p(q) \le p_0 \\ 0 & p(q) > p_0 \end{cases} \tag{23}$$

Among them,, $U_{ATT}(Q)$ represents the value of the attractive field, $P_G(Q)$ represents the distance to the target position, and k represents the attraction gain constant. $U_{REP}(Q)$ represents the value of the repulsive field, P(Q) represents the distance to obstacles, η represents the repulsion gain constant, and P_0 represents the influence range of obstacles.

The direction of motion is generated by taking the negative gradient of the force field function, as shown below:

$$F_{att}(q) = -kp_G(q) \tag{24}$$

$$F_{rep}(q) = \begin{cases} \eta\left(\dfrac{1}{p(q)} - \dfrac{1}{p_0}\right)\cdot\dfrac{\nabla p(q)}{p^2(q)} & p(q) \le p_0 \\ 0 & p(q) > p_0 \end{cases} \tag{25}$$

$F_{ATT}(Q)$ represents attraction in the above formula, and $F_{REP}(Q)$represents repulsion.

The overall field combines an attraction field and a repulsion field. Taking the gradient of the resultant field provides the direction in which an object should move, as shown below:

$$U(q) = U_{att}(q) + U_{rep}(q) \tag{26}$$

$$F(q) = -\nabla U(q) \tag{27}$$

The paths generated by the artificial potential field method are generally smooth and safe, but this approach suffers from the local optima problem.

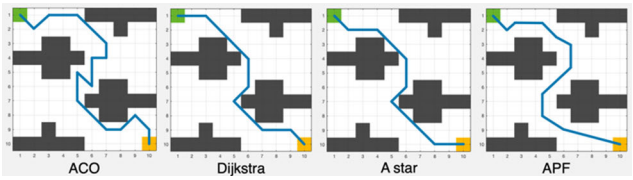

FIGURE 11. ACO, Dijkstra, A star, APF Algorithm in Grid-2 Search Path.

TABLE 4. Compare the experimental data of the algorithm.

Algorithm	Single search time	Step	Number of inflection points
ACO	0.68 s	22.7	13
Dijkstra	0.04 s	15.9	6
A star	0.02 s	15.9	6
APF	0.30 s	17.2	9
QL-DYN	0.58 s	20	8

For comparative algorithm experiments, the Grid-2 map with moderate obstacle complexity from Test Case III-B was chosen as the map model. A typical starting point (1, 1) and ending point (10, 10) were used. The diagram below depicts the paths searched by the four comparative algorithms in Grid-2. In ACO, Dijkstra, and A* algorithms, the search directions were uniformly set to eight directions. The smoothed paths of the Artificial Potential Field (APF) method were plotted as line segments based on their directional characteristics to compare the search trajectories of different algorithms visually.

From the above data, it can be observed that among the five comparative algorithms, Dijkstra and A* demonstrate superior efficiency. A* combines Dijkstra's principles and selects nodes most likely to lead to the shortest path based on a heuristic function estimating the distance to the goal node and the actual path taken. This approach efficiently searches the space and is usually faster than Dijkstra. Compared to the above two methods, Q-learning can learn in complex dynamic environments, adapt to changing weights and environmental conditions, and gradually optimize strategies through interaction with the environment. Traditional Dijkstra and A* algorithms, on the other hand, are typically suited for static environments. Ant Colony Optimization (ACO) often requires ants to share pheromone information and coordinate actions, which may necessitate global information.

In contrast, the Q-learning algorithm can learn and make decisions based on local states. The artificial potential field method is prone to local minima, especially when there are intricate obstacles. Q-learning, employing explorationexploitation strategies, is better equipped to avoid such issues.

It is important to note that despite having many advantages, Q-learning also has some limitations, such as requiring a significant amount of iterative training and time, and dependence on the initial state space. Choosing the appropriate algorithm depends on the characteristics and requirements of the problem. In practical applications, it is necessary to select different algorithms or combine them based on the specific features of the problem.

C. AIRPORT TAXIWAY COLLISION AVOIDANCE STRATEGY

1) OPERATIONAL RULES

Due to the specificity of aircraft operations, aircraft pilots are required to strictly adhere to the following safety taxiing regulations when taxiing or towing on the ground:

- If two aircraft approach each other head-on, they must each move to the right and maintain a safe separation from each other.
- When two aircraft cross each other, the pilot of the aircraft who sees the other aircraft on his left side from the cockpit must stop taxiing and yield.
- When two aircraft are taxiing in a follow-the-leader manner, the trailing aircraft must not overtake the leading aircraft and should maintain a safe distance of no less than 50 meters.
- Aircraft are not allowed to cross a runway during taxiing without permission from the airport control.

2) COLLISION AVOIDANCE STRATEGY

Based on the listed airport taxi control regulations above, we introduced corresponding avoidance strategies and constraints in the dynamic programming experiments in Chapter 5.

- Strategy 1: Airport taxiways typically follow a one-way, forward principle to ensure the safety and efficiency of aircraft operations. Therefore, controllers usually do not allow two aircraft to taxi toward each other. In our algorithm, we incorporate a decision to terminate the iteration search when the Agent and a dynamic obstacle move in opposite directions and meet at the exact location in the two-dimensional space.
- Strategy 2: We introduce a check in the experiment to address the priority issue when two aircraft meet at an intersection. Before each movement, the Agent will check if there is a dynamic obstacle in the left front of the intended direction of movement. If a dynamic obstacle is detected, the Agent will pause the search for three steps, allowing the prioritized Agent or dynamic obstacle to pass and maintain a safe distance before continuing the search.
- Strategy 3: In the simulated airport grid environment, the Agent and dynamic obstacles are set to have the same speed, and the width of the taxiway occupies one grid unit. Therefore, the Agent and dynamic obstacles may follow each other during their movement. In the experiment, we set up three equally-sized and independent dynamic obstacles. By defining the taxiing routes of the three dynamic obstacles, we create a safe margin of 50m $\times$50m grid units between the two aircraft for follow-up taxiing and safe intersection encounters.
- Strategy 4: To prevent the Agent from crossing the runway during the experiment, we turn off the grid areas

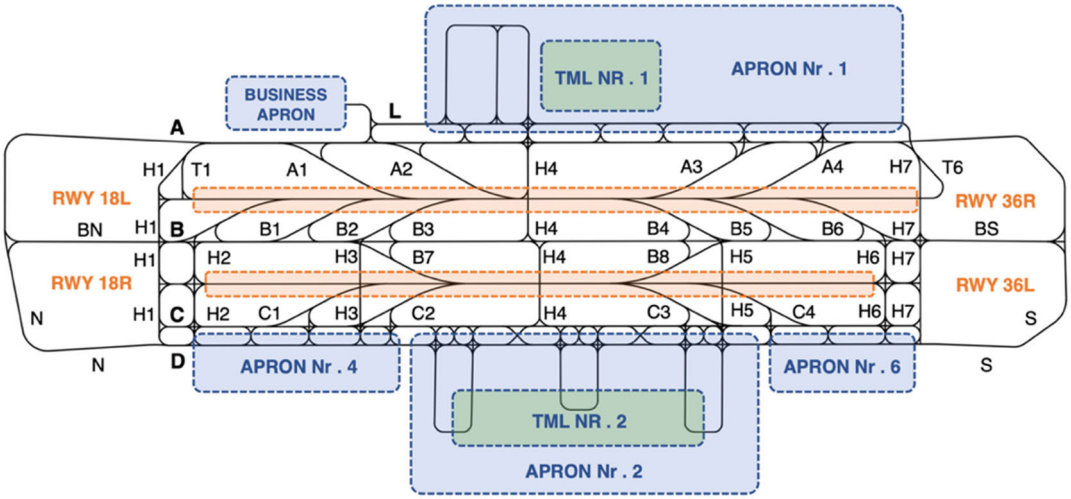

FIGURE 12. Shanghai Hongqiao Airport (ZSSS) structural diagram (The blue area is the apron area, the green area is the terminal area, and the orange area is the runway.)

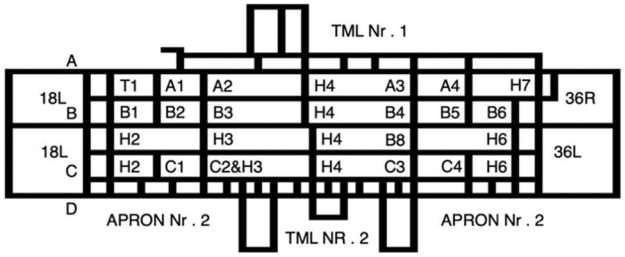

FIGURE 13. Airport environment grid representation diagram.

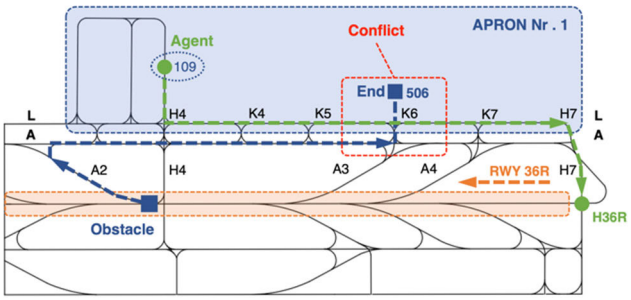

FIGURE 14. Simulation taxiway route diagram: departure using runway 36R.

corresponding to the runways, except for the taxiways necessary for aircraft to depart from the runway.

In conclusion, under the premise of the Agent's path search objective function in Chapter 3, the position constraints after introducing dynamic obstacles can be represented as follows:

$$P_a = (x_i \pm 1, y_i \pm 1) \in (X,Y) \tag{28}$$

$$P_{ob} = (x_j \pm 1, y_j \pm 1) \in (X,Y) \tag{29}$$

$$(x_i \pm 1, y_i \pm 1) \neq (x_j \pm 1, y_j \pm 1) \tag{30}$$

$$|x_i - x_j| + |y_i - y_j| \geq 1 \tag{31}$$

In the above formula, P_a represents the position state of the Agent, and P_{ob} represents the position state of the dynamic obstacle.

V. SIMULATION AND RESULTS

A. SIMULATION ENVIRONMENT

In this section, the improved algorithm approach from the previous chapter is applied to the Shanghai Hongqiao Airport simulation environment to evaluate the established dynamic conflict avoidance model. With its narrow parallel runways, Shanghai Hongqiao International Airport serves as a typical international airport and possesses a certain degree of universality. In most cases, it can represent operational scenarios in airports worldwide. Within the established grid model, obstacles are categorized into two types: static obstacles and dynamic obstacles. Static obstacles represent areas where aircraft movement is prohibited, while dynamic obstacles are used to simulate other aircraft or vehicles currently operating on the airport apron. The movement of dynamic obstacles can be customized by defining individual step-wise motion nodes.

As shown in Figure 12, Shanghai Hongqiao Airport has two parallel narrow runways in the north-south direction: Runway 36 and Runway 18, both available for aircraft takeoff and landing. Six rapid exit taxiways are connected to the runways: C1, C2, C3, C4, B7, and B8. On the west side of these taxiways are connected to the second, fourth, and sixth aprons. Runway 36R - 18L has a total of ten rapid exit taxiways: A1, A2, A3, A4, B1, B2, B3, B4, B5, and B6. On the east side of these taxiways are connected to the first apron and the business apron.

As shown in Figure 13, we modeled the runway, taxiway, and apron areas of Shanghai Hongqiao Airport for conducting dynamic aircraft operation simulation experiments. To simplify the simulation environment model structure, we gratified it and established a grid map of 50·80, where

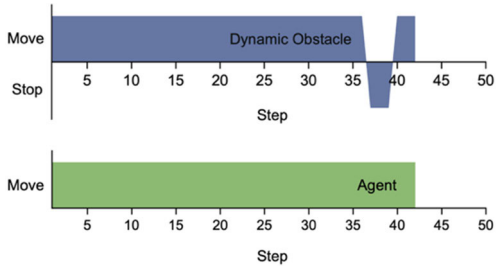

FIGURE 15. Agent and dynamic obstacle state during operation(36R-D).

TABLE 5. Taxiing routes of intelligent agents and dynamic obstacles: departure using runway 36R.

Object	Taxiing Routes
Dynamic Obstacle	RWY 36R-A2-A-K6-506
Agent	109-H4-L-H7-H36R

each grid represents an actual area of 50m·50m. The relative positions for fast exiting from and connecting parallel taxiways A1, A2, A3, A4, B1, B2, B3, B4, B5, B6, B7, B8, C1, C2 were used as the reference.

We conducted four sets of simulation experiments, each reflecting the situations of two active runways at Shanghai Hongqiao Airport, involving different directions for arrivals and departures. Among these, scenario 36R-A was divided into three groups based on whether aircraft can cross another runway during taxiing, following airport operational regulations. These considerations encompass all typical operational states of the airport and hold a certain level of generality among international airports worldwide.

The movement routes of dynamic obstacles were calibrated based on airport taxiing regulations, and they started searching and moving at the same time as the Agent, with the same movement speed. As for the parameters, we set the Max episode to 100, $\varepsilon = 0.1$, Dynamic-$\varepsilon = 0.004$, $\alpha = 0.1$, $\gamma = 0.9$, and Initial $Q = 1/d_m$.

B. SIMULATION EXPERIMENT

1) RWY-36R-DEPARTURE

In this set of simulation experiments, we use 36R as the active runway. We set dynamic obstacles to taxi from runway 36R, depart from taxiway A2, and taxi to parking stand 506, located in the apron area of gate 1. At the same time, we ask the Agent to search for the optimal taxiing route from parking stand 109 to the holding point outside runway 36R. The critical point is that the Agent cannot traverse the track during the search process, so we have closed the entrances of taxiways A2, H4, A3, and A4. The results and the state of the Agent are shown in Figure 14.

As shown in Figure 15, during the operation, we observed a crossing conflict between the Agent and the dynamic obstacle near taxiways K5 and K6 between steps 35 and 40. At this time, the Agent was positioned to the left and in front of the dynamic obstacle, giving it a priority of passage.

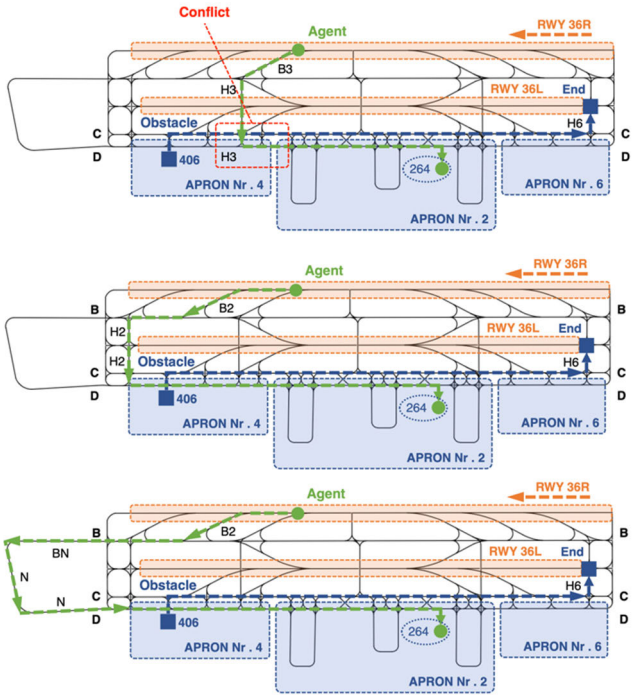

FIGURE 16. Simulation taxiway route diagram: arrival using runway 36R.

2) RWY-36R-ARRIVAL

We use runways 36R and 36L as the operating runways. Dynamic obstacles are set to taxi from parking stand 406, located in apron 4, via taxiways C and H6 to the holding point outside runway 36L. At the same time, the Agent is tasked with searching for the optimal taxi route from the departure point on runway 36R to parking stand 26, located in Apron 2. It is important to note that there is no rapid exit taxiway on the west side of runway 36R at Shanghai Hongqiao Airport. Only three perpendicular taxiways, B1, B2, and B3, are available for aircraft to exit southwards. Among them, B3 crosses the northern area of runway 36L. Aircraft can only exit westwards from B3 when permitted by the tower control. Therefore, we conducted three of simulation experiments: B3-H3 available, B3-H3 unavailable, and B3-H3-H2 unavailable. The results of the experiments are shown in the following figure.

Figures 16 and 17 show that when the aircraft departs from B1 and B2, there is no spatial conflict with the dynamic obstacles. However, when the aircraft departs from B3, a crossing conflict occurs at the intersection of H3 and D taxiways. In this case, the Agent is positioned to the left of the dynamic obstacle and has the right of way. Therefore, the Agent proceeds from H3 to D while the dynamic obstacle moves south. The Agent proceeds southward and reaches parking position 264.

3) RWY-18R-DEPARTURE

We use runway 18R for operation. Dynamic obstacles are set to taxi from C3 through H5 to parking stand 229, located at

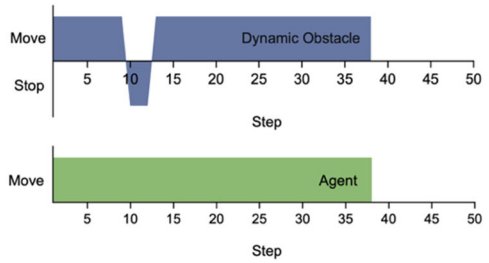

FIGURE 17. Agent and dynamic obstacle state during operation(36R-A).

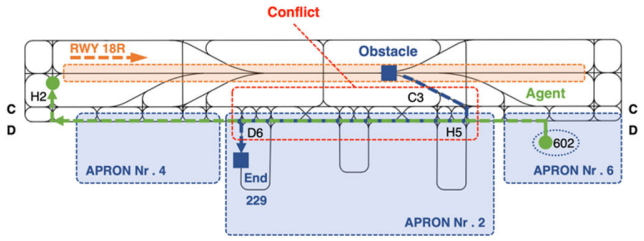

FIGURE 18. Simulation taxiway route diagram: departure using runway 18R.

TABLE 6. Taxiing routes of intelligent agents and dynamic obstacles arrival using runway 36R.

Object	Taxiing Routes
Dynamic Obstacle	406-D-D3-C-H6-H36L
Agent-a	RWY 36R -B3-H3-D-264
Agent-b	RWY 36R -B2-B-H2-D-264
Agent-c	RWY 36R -B1-BN-N-D-264

TABLE 7. Taxiing routes of intelligent agents and dynamic obstacles: departure using runway 36R.

Object	Taxiing Routes
Dynamic Obstacle	RWY 18R-C3-H5-C-D6-229
Agent	602-D-H2-H18R

apron 2. At the same time, we instruct the Agent to search for the optimal taxi route from parking stand 602 at Apron 6 to the holding point outside runway 18R. Similarly, the Agent is not allowed to cross the runway during the search process, so we close the entrances of taxiways C1, C2, H4, C3 (H5), and C4. The operational results are shown in the following figure:

We observe that when the Agent and dynamic obstacle approach H5, they enter a scenario of following each other's taxi path. At this point, the dynamic obstacle appears in front of the Agent, approximately three grid cells away (around 150 meters) from its current position. As the dynamic obstacle reaches D6 and turns towards M2, the Agent continues moving northward.

4) RWY-18R-ARRIVAL

In this set of simulation experiments, we use runway 18R for operations. Dynamic obstacles are set to taxi from parking stand 602, located at apron 6, through taxiways D and H2,

TABLE 8. Taxiing routes of intelligent agents and dynamic obstacles: arrival using runway 18R.

Object	Taxiing Routes
Dynamic Obstacle	602-D-H2-H18R
Agent	RWY 18R-C3-C-H3-229

TABLE 9. Time performance improvement data.

Algorithm Model	QL	QL*	Improvement (time)
RWY-36R-D	78	69	11.5%
RWY-36R-A-a	106	80	24.5%
RWY-36R-A-b	146	121	17.1%
RWY-36R-A-c	227	172	24.2%
RWY-18R-D	36	33	8.3%
RWY-18R-A	141	127	9.9%

TABLE 10. Convergence performance improvement data.

Algorithm Model	QL	QL*	Improvement (convergence)
RWY-36R-D	56	53	5.4%
RWY-36R-A-a	78	58	25.6%
RWY-36R-A-b	94	66	29.8%
RWY-36R-A-c	91	69	24.2%
RWY-18R-D	41	40	2.5%
RWY-18R-A	95	83	12.6%

to the holding point outside runway 18R. Meanwhile, the Agent is tasked with searching for the optimal taxi route from taxiway C3 to parking stand 414, located at apron 4. The operational results are shown in the following figure:

As shown in Figures 19 and 20, during the operation, we observed a crossing conflict when the Agent and the dynamic obstacle approached the vicinity of H3. The dynamic obstacle was placed to the front left of the Agent at this time and had the right-of-way. Therefore, the Agent paused its search for three steps and waited for the dynamic obstacle to pass before continuing its westward search.

C. RESULTS

In the same simulation environment, we search for the optimal taxi routes for multiple departing and arriving aircraft groups. To demonstrate the improvement of the proposed algorithm, we also perform path searches using the original Q-Learning algorithm under the same parameter conditions in each group experiment. The table below shows the search time and convergence iteration count for the original algorithm (QL) and the improved algorithm (QL*).

The improved algorithm shows significant improvements in search time and convergence speed. For example, in the case of RWY-36R-A, compared to the traditional Q-Learning algorithm, the improved algorithm reduces the search time by

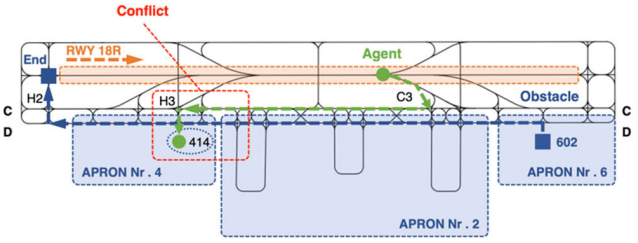

FIGURE 19. Simulation taxiway route diagram: arrival using runway 18R.

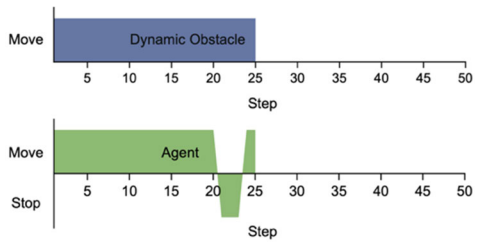

FIGURE 20. Agent and dynamic obstacle state during operation(18R-A).

24.2%, decreases the convergence search count from 94 to 66, and achieves a 29.8% improvement in convergence performance. This can be attributed to the exploration guidance and the initial values in the Q-Table. Introducing of the dynamic exploration factor eliminates the instability caused by the exploration rate in the later stages of the algorithm, leading to a substantial increase in search speed.

VI. CONCLUSION

Providing taxi route guidance for aircraft on the apron is the most fundamental means of airport control. We propose improving the Q-Learning algorithm in reinforcement learning to provide taxi route guidance for aircraft. First, we establish a grid-based airport environment model for Shanghai Hongqiao Airport in China. Then, we make various improvements to the traditional Q-Learning algorithm in reinforcement learning, including optimizing the exploration strategy and resetting Q-values to enhance algorithm efficiency. We also introduce a dynamic exploration factor to improve the algorithm's stability during later convergence. Based on civil airports' basic apron control rules, we incorporate corresponding conflict avoidance strategies for aircraft in the algorithm. This method can be well integrated into the path planning algorithm and, compared to the original Q-Learning algorithm, can accurately and quickly find the shortest taxi route for arriving and departing aircraft in compliance with the apron operational regulations. Finally, the proposed approach is validated at Shanghai Hongqiao Airport.

REFERENCES

[1] A. E. I. Brownlee, M. Weiszer, J. Chen, S. Ravizza, J. R. Woodward, and E. K. Burke, "A fuzzy approach to addressing uncertainty in airport ground movement optimisation," *Transp. Res. C, Emerg. Technol.*, vol. 92, pp. 150–175, Jul. 2018, doi: 10.1016/j.trc.2018.04.020.

[2] L. Hao, M. S. Ryerson, L. Kang, and M. Hansen, "Estimating fuel burn impacts of taxi-out delay with implications for gate-hold benefits," *Transp. Res. C, Emerg. Technol.*, vol. 80, pp. 454–466, Jul. 2017, doi: 10.1016/j.trc.2016.05.015.

[3] G. Clare and A. G. Richards, "Optimization of taxiway routing and runway scheduling," *IEEE Trans. Intell. Transp. Syst.*, vol. 12, no. 4, pp. 1000–1013, Dec. 2011, doi: 10.1109/TITS.2011.2131650.

[4] C. Evertse and H. G. Visser, "Real-time airport surface movement planning: Minimizing aircraft emissions," *Transp. Res. C, Emerg. Technol.*, vol. 79, pp. 224–241, Jun. 2017, doi: 10.1016/j.trc.2017.03.018.

[5] E. I. Brownlee, J. R. Woodward, M. Weiszer, and J. Chen, "A rolling window with genetic algorithm approach to sorting aircraft for automated taxi routing," in *Proc. Genetic Evol. Comput. Conf.*, Kyoto, Japan, Jul. 2018, pp. 1207–1213, doi: 10.1145/3205455.3205558.

[6] W. Deng, L. Zhang, X. Zhou, Y. Zhou, Y. Sun, W. Zhu, H. Chen, W. Deng, H. Chen, and H. Zhao, "Multi-strategy particle swarm and ant colony hybrid optimization for airport taxiway planning problem," *Inf. Sci.*, vol. 612, pp. 576–593, Oct. 2022, doi: 10.1016/j.ins.2022.08.115.

[7] J. Atkin, E. K. Burke, and S. Ravizza, "The airport ground movement problem: Past and current research and future directions," in *Proc. 4th Int. Conf. Res. Air Transp.*, Budapest, Hungary, 2010, pp. 131–138.

[8] T. Zhang, M. Ding, B. Wang, and Q. Chen, "Conflict-free time-based trajectory planning for aircraft taxi automation with refined taxiway modeling," *J. Adv. Transp.*, vol. 50, no. 3, pp. 326–347, Apr. 2016, doi: 10.1002/atr.1324.

[9] S. J. Landry, X. W. Chen, and S. Y. Nof, "A decision support methodology for dynamic taxiway and runway conflict prevention," *Decis. Support Syst.*, vol. 55, no. 1, pp. 165–174, Apr. 2013, doi: 10.1016/j.dss.2013.01.016.

[10] Y. F. Chen, M. Liu, M. Everett, and J. P. How, "Decentralized non-communicating multiagent collision avoidance with deep reinforcement learning," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, Singapore, May 2017, pp. 285–292, doi: 10.1109/ICRA.2017.7989037.

[11] M. Everett, Y. F. Chen, and J. P. How, "Motion planning among dynamic, decision-making agents with deep reinforcement learning," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst. (IROS)*, Madrid, Spain, Oct. 2018, pp. 3052–3059, doi: 10.1109/IROS.2018.8593871.

[12] J. P. Bailey, A. Nash, C. A. Tovey, and S. Koenig, "Path-length analysis for grid-based path planning," *Artif. Intell.*, vol. 301, Dec. 2021, Art. no. 103560, doi: 10.1016/j.artint.2021.103560.

[13] F. Kiani, A. Seyyedabbasi, S. Nematzadeh, F. Candan, T. Çevik, F. A. Anka, G. Randazzo, S. Lanza, and A. Muzirafut, "Adaptive metaheuristic-based methods for autonomous robot path planning: Sustainable agricultural applications," *Appl. Sci.*, vol. 12, p. 943, Jan. 2022, doi: 10.3390/app12030943.

[14] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, Feb. 2015, doi: 10.1038/nature14236.

[15] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis, "Mastering the game of go with deep neural networks and tree search," *Nature*, vol. 529, no. 7587, pp. 484–489, Jan. 2016, doi: 10.1038/nature16961.

[16] C. Amato and G. Shani, "High-level reinforcement learning in strategy games," in *Proc. 9th Int. Conf. Autonomous Agents Multiagent Syst.*, Toronto, ON, Canada, 2010.

[17] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," in *Proc. 4th Int. Conf. Learn. Represent.*, San Juan, Puerto Rico, 2016.

[18] M. Brittain and P. Wei, "Autonomous aircraft sequencing and separation with hierarchical deep reinforcement learning," in *Proc. Int. Conf. Res. Air Transp.*, Castelldefels, Spain, Jun. 2018.

[19] M. Brittain and P. Wei, "Autonomous air traffic controller: A deep multi-agent reinforcement learning approach," in *Proc. 36th Int. Conf. Mach. Learn.*, California, CA, USA, 2019.

[20] C. J. C. H. Watkins, "Learning from delayed rewards," *Robot. Auton. Syst.*, vol. 15, no. 4, pp. 233–235, 1989, doi: 10.1016/0921-8890(95)00026-C.

[21] F. Moreno-Vera, "Performing deep recurrent double Q-learning for Atari games," in *Proc. IEEE Latin Amer. Conf. Comput. Intell. (LA-CCI)*, Nov. 2019, pp. 1–4, doi: 10.1109/LA-CCI47412.2019.9036763.
[22] R. S. Sutton and A. G. Barto, "Reinforcement learning: An introduction," *IEEE Trans. Neural Netw.*, vol. 9, no. 5, p. 1054, Sep. 1998, doi: 10.1109/TNN.1998.712192.
[23] L. Martin, *Markov Decision Processes: Discrete Stochastic Dynamic Programming*, vol. 37. Hoboken, NJ, USA: Wiley, Apr. 1994, doi: 10.1002/9780470316887.
[24] J. Filar and K. Vrieze, *Competitive Markov Decision Processes*. Berlin, Germany: Springer-Verlag, Dec. 1996, p. 393.
[25] J. Schulman, S. Levine, P. Moritz, M. Jordan, and P. Abbeel, "Trust region policy optimization," in *Proc. 32nd Int. Conf. Int. Conf. Mach. Learn.*, vol. 37, Jul. 2015, pp. 1889–1897.
[26] 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. 39974–39982, 2019, doi: 10.1109/ACCESS.2019.2902846.
[27] C. J. C. H. Watkins and P. Dayan, "Q-learning," *Mach. Learn.*, vol. 8, nos. 3–4, pp. 279–292, May 1992, doi: 10.1007/BF00992698.
[28] B. Luo, D. Liu, T. Huang, and D. Wang, "Model-free optimal tracking control via critic-only Q-learning," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 27, no. 10, pp. 2134–2144, Oct. 2016, doi: 10.1109/TNNLS.2016.2585520.
[29] Y. Freund and R. E. Schapire, "A decision-theoretic generalization of on-line learning and an application to boosting," in *Proc. Eur. Conf. Comput. Learn. Theory*. Berlin, Germany: Springer, 1997, pp. 119–139.
[30] A. Al-Tamimi, F. L. Lewis, and M. Abu-Khalaf, "Discrete-time nonlinear HJB solution using approximate dynamic programming: Convergence proof," *IEEE Trans. Syst., Man, Cybern., B, Cybern.*, vol. 38, no. 4, pp. 943–949, Aug. 2008, doi: 10.1109/TSMCB.2008.926614.
[31] A. Ammar, H. Bennaceur, I. Châari, A. Koubâa, and M. Alajlan, "Relaxed Dijkstra and A with linear complexity for robot path planning problems in large-scale grid environments," *Soft Comput.*, vol. 20, no. 10, pp. 4149–4171, Oct. 2016, doi: 10.1007/s00500-015-1750-1.
[32] R. Cui, Y. Li, and W. Yan, "Mutual information-based multi-AUV path planning for scalar field sampling using multidimensional RRT," *IEEE Trans. Syst., Man, Cybern., Syst.*, vol. 46, no. 7, pp. 993–1004, Jul. 2016, doi: 10.1109/TSMC.2015.2500027.
[33] S. M. Lavalle, "Rapidly-exploring random trees: A new tool for path planning," *IEEE Trans. Robot. Autom.*, vol. 15, no. 5, pp. 990–1007, Jan. 1998.
[34] Z. Shi, J. Tu, Q. Zhang, X. Zhang, and J. Wei, "The improved Q-learning algorithm based on pheromone mechanism for swarm robot system," in *Proc. 32nd Chin. Control Conf.*, Xi'an, China, Jul. 2013, pp. 6033–6038.
[35] M. Dorigo, V. Maniezzo, and A. Colorni, "Ant system: Optimization by a colony of cooperating agents," *IEEE Trans. Syst., Man, Cybern., B, Cybern.*, vol. 26, no. 1, pp. 29–41, Feb. 1996, doi: 10.1109/3477.484436.
[36] E. W. Dijkstra, "A note on two problems in connexion with graphs," *Numerische Math.*, vol. 1, no. 1, pp. 269–271, Dec. 1959, doi: 10.1007/bf01386390.
[37] P. E. Hart, N. J. Nilsson, and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," *IEEE Trans. Syst. Sci. Cybern.*, vol. SSC-4, no. 2, pp. 28–29, Jul. 1968, doi: 10.1145/1056777.1056779.
[38] O. Khatib, "Real-time obstacle avoidance for manipulators and mobile robots," *Int. J. Robot. Res.*, vol. 5, no. 1, pp. 500–505, Feb. 1985, doi: 10.1109/robot.1985.1087247.

ZHENG XIANG (Member, IEEE) received the B.S. degree in engineering and the Ph.D. degree in communication and information systems from Southwest Jiaotong University, in 2005 and 2011, respectively. He is currently a Teacher with the Air Traffic Control Department, Air Traffic Control College, Civil Aviation Flight University of China, and is a Master's Student Supervisor. From 2014 to 2015, he was with the Airspace Management Center, Civil Aviation Administration of China (CAAC). In 2016, he went to Sweden to attend a three-month refresher training for air traffic control teachers at the Nordic Institute of Air Traffic Control. His main research interests are in air traffic management automation. He has published over 20 articles, one textbook, three patents, and software copyrights and has hosted and participated in more than ten teaching and research projects.

HEYANG SUN (Member, IEEE) received the B.S. degree in engineering from the School of Transportation Engineering, Zhejiang College, Tongji University, in 2021. He is currently pursuing the master's degree with the Civil Aviation Flight University of China, College of Air Traffic Management. His main research interests include machine learning and air traffic management automation.

JIAHAO ZHANG received the B.S. degree in engineering from the Jincheng College, Nanjing University of Aeronautics and Astronautics, in 2022. He is currently pursuing the master's degree in transportation with the College of Air Traffic Management, Civil Aviation Flight University of China.