International Journal of Innovative Computing, Information and Control
Volume **18**, Number **3**, June **2022**
ICIC International ©2022 ISSN 1349-4198
pp. **687–702**

PATH PLANNING OF MOBILE ROBOT BASED ON THE IMPROVED Q-LEARNING ALGORITHM

Chaorui Chen and Dongshu Wang
School of Electrical Engineering
Zhengzhou University
No. 100, Science Road, Zhengzhou 450001, P. R. China
872109865@qq.com; wangdongshu@zzu.edu.cn

Received October 2021; revised February 2022

Abstract. *For path planning of mobile robots in static environment, the traditional Q-learning algorithms suffer from slow convergence speed, difficulty in converging to the optimal solution, and poor generalization ability, etc. To solve this problem, an improved Q-learning algorithm is presented. For the slow convergence, this paper 1) modifies the discount rate, learning rate and other parameters to promote the accuracy of value updating; 2) introduces single-chain backtracking algorithm to improve the learning speed of agent; 3) designs Q-learning algorithm based on single-chain set to solve the invalid state loops in single-chain; 4) combines the artificial potential field method and the shared single-chain library to enhance the target guidance in the environment exploration of the agent. To solve the problem that the agent converges too fast in some specific states and may converge to non-optimal solutions, a novel exploration rate is designed, and the reward function is modified. To improve the generalization ability of the traditional Q-learning algorithm, a new Dyna-2 algorithm is designed. Finally, the potential of the improved Q-learning algorithm proposed is verified by comparison experiments under four simulation environments.*

Keywords: Path planning, Q-learning, Generalization capacity, Mobile robot

1. Introduction.

Path planning is an important research direction of mobile robot and the key to achieving the navigation tasks, that is, mobile robots can independently explore a smooth and collision-free path trajectory from the initial position to the target position.

Traditional path planning algorithms include A* algorithm [1], artificial potential field method [2], Dijkstra algorithm [3], fast expanding random tree method [4], etc. These algorithms are used to solve the path planning in known environment and are easy to implement, but robots have poor exploration ability in path planning, and they are easy to fall into the local optimum.

According to the training methods, the path planning of mobile robot can be divided into supervised learning, unsupervised learning and reinforcement learning (RL). Both the supervised learning and the unsupervised learning need a large amount of prior knowledge. Otherwise, it is not possible for them to conduct a good path planning. Reinforcement learning is an artificial intelligence algorithm that does not need prior knowledge, but directly carries out trial-and-error in the environment to obtain feedback to optimize the strategy. It is also capable of autonomous learning and online learning. Therefore, the RL has gradually become a research hotspot in path planning in unknown environment of the mobile robots [5-8].

As a classical representation of the RL, Q-learning has achieved wide applications in the path planning of the mobile robots. For example, for the obstacle avoidance of an

DOI: 10.24507/ijicic.18.03.687

autonomous quadrotor, Ou et al. [9] used a dueling double deep recurrent Q-learning to eliminate the adverse effects of the on-board monocular camera's limited observation capacity while choosing practical obstacle avoidance action. Li et al. [10] used a deep Q-learning network (DQN) to continuously interact with the visually simulated environment to obtain experience data, so that the agent can learn the best action strategies in the visual simulated environment. Moreover, the artificial potential field (APF) algorithm is adopted to promote the action space and reward function of the DQN algorithm. Kontoudis and Vamvoudakis [11] proposed an online kinodynamic motion planning framework with asymptotically optimal rapidly-exploring random tree and continuous-time Q-learning. A model-free Q-based advantage function was formulated and integral RL was used to develop tuning laws for the online approximation of the optimal cost and the optimal policy of a mobile robot. Jin et al. [12] designed an event-driven recurrent Q-learning to focus on the motion planning of agents towards intersection scenarios to conclude a sample path with safety and efficiency. Jiang et al. [13] designed a novel approach based on deep Q-learning with experience replay and heuristic knowledge, to address the path planning and obstacle avoidance of the intelligent robots. A neural network was used to resolve the "curse of dimensionality" problem of the Q-table in RL. Heuristic knowledge was used to help the robot to avoid blind exploration and provide more effective data for training the neural network.

Though the Q-learning algorithm has achieved wide applications in the path planning of the mobile robots, there are still some drawbacks existing in the traditional Q-learning algorithm, such as slow convergence speed, long planned path, and poor generalization capacity. Many researchers have attempted different methods to solve these problems [14-18]. Although these methods have achieved certain performance, unfortunately, they only address one or two of the drawbacks well. For example, Yao et al. [17] combined improved black-hole potential field and reinforcement learning to solve the local minimum problem, without considering other drawbacks. Zhao et al. [18] proposed the experience-memory Q-learning algorithm to deal with the problem of slow convergence speed and long planned path, and due to the experience-memory table, its generalization ability is limited. Therefore, new algorithms are still needed to solve these drawbacks simultaneously.

Bearing this in mind, this paper proposed a novel improved Q-learning algorithm to solve the slow convergence speed, and poor generalization capacity. Firstly, in order to promote the convergence speed of the traditional Q-learning algorithm, this work 1) updates the learning rate and discount rate, and introduces the single-chain backtracking algorithm; 2) develops Q-learning algorithm based on single-chain set to deal with the invalid state loops in single-chain; 3) designs an algorithm to share a single-chain library under multiple training episodes; 4) introduces the APF method, and integrates the approach based on the shared single-chain library to achieve sub-goals, to promote the target guidance in the environment exploration of the robots. Secondly, to solve the drawback that the robot converges too fast in some specific states and possibly converges to local optimum, a novel exploration rate based on the number of paths is designed, and a reward function is designed. Thirdly, to improve the generalization ability of the traditional Q-learning algorithm, a new Dyna-2 algorithm is designed. Finally, four simulations are designed to demonstrate the potential of the improved Q-learning algorithm proposed in this paper.

Section 2 explains the theory of the single-chain backtracking Q-learning algorithm. Section 3 and Section 4 design the Q-learning algorithm based on the single-chain set and the SSQ-learning with APF algorithm, respectively. Section 5 and Section 6 explain how to improve performance based on the path number and the generalization ability of traditional Q-learning algorithm, respectively. Section 7 conducts the simulation experiment and analyzes the results. Section 8 concludes the research and points out the future research topics.

2. Single-Chain Backtracking Q-Learning Algorithm.

Single-chain backtracking Q-learning algorithm is based on the idea of backtracking and single-chain. The purpose of this algorithm is to improve the convergence speed of the reinforcement learning. According to the characteristics of different states with different convergence priorities, it can make the states near the target to converge earlier, provide necessary conditions for the convergence of other states, improve the lag of Q-learning through the idea of backtracking, and update more values at a time.

The state experienced by the robot in the process of path planning from the start state to a certain state is regarded as a chain. The start state is the first element of the single-chain, the next state of the start state is the second element of the single-chain, and so on. Until reaching the current state of the agent, the formed state chain is called a single-chain.

The single-chain backtracking algorithm uses the single-chain to transfer and backtrack data. Through the circular recursion, the influence of a decision is iteratively updated to the previous state, so as to avoid meaningless search and speed up the convergence rate.

2.1. Algorithm description.

The core of the single-chain backtracking algorithm is to introduce the memory function. By recording the relevant parameters, Q value of the recorded state-action pair is updated iteratively during the operation of the agent, so as to accelerate the convergence rate. Memory matrix can be defined as follows.

Definition 2.1. *$M(t) \leftarrow (s, a, r, \lambda)$, where s is the state of the agent at time t, a is the corresponding action decision, r is the corresponding reward obtained by the agent taking action a in this state, and λ is the learning rate.*

In single-chain backtracking algorithm, the required data are recorded into the memory matrix according to the running state of the agent, and then the Q value is updated according to the memory matrix. The update formula is denoted as follows:

```
for $k = t-1, t-2, \ldots, 2, 1$
    $Q_{t+1}(s_k, a_k) \leftarrow (1-\lambda_k)Q_t(s_k, a_k) + \lambda_k(r_k + \gamma^* \max_{a_t \in A} Q_{t+1}(S_{k+1}, a_k))$
until $k = 1$
end
```

In updating the Q value corresponding to the latest state-action pair at time t, the previously recorded states in the single-chain will be updated together, so as to obtain a higher convergence speed. This algorithm will become the basis for calculating Q value in Section 3.

2.2. Setting the learning rate.

Considering that in daily learning activities, if people repeat learning the same knowledge, the amount of knowledge obtained by repeated learning is actually decreasing because they already know the knowledge. Therefore, the learning rate is set as a function that decreases with the increase of the training episodes:

$$\lambda(s, a) = \lambda_a / \sqrt[5]{episodes} \tag{1}$$

where λ_a is a constant. When the training episodes *episodes* increase, the learning rate decreases gradually, and the Q value tends to maintain the original value and accelerate the convergence speed of Q value.

3. **Q-Learning Algorithm Based on Single-Chain Set.**

In the above single-chain backtracking algorithm, the state chain stored in the single-chain has not been processed, which will produce a state loop composed of many repeatedly experienced invalid states. For example, even if there are only three states (s_1 start state, s_2 intermediate state, s_3 target state), the following may occur:

$s_1 \rightarrow s_2 \rightarrow s_1 \rightarrow s_2 \rightarrow s_1 \rightarrow s_2 \rightarrow s_1 \rightarrow s_2 \rightarrow s_1 \rightarrow s_2 \rightarrow s_3$

If the single-chain backtracking algorithm is still used for direct calculation, it may affect the convergence speed of Q value in some cases. Therefore, a Q-learning algorithm based on single-chain set is designed to solve the state loop and update the Q value.

Definition 3.1. *In a training episode, the agent starts from state s_0 to target state s, adds all experienced state s in reverse order into a single-chain, and the resulting single-chain is defined as state-chain(s, s_0), which records the states that the agent has experienced from the initial state s_0 to the target state s in reverse order. The single-chain set is generated from the state chain. Each single-chain in the state-set is the shortest acyclic state path from one state to the target state s, the single-chain set contains all the shortest acyclic state paths that can be found from the state chain, and the single-chain set is denoted as state-set(s).*

When the Q value of state s is updated, the update can be propagated to all the states the agent has experienced in reverse order along each single-chain stored in the single-chain set, which not only solves the problem that the traditional Q-learning algorithm has too few Q values to update in one iteration, but also solves the problem of invalid state loops in the single-chain backtracking algorithm.

3.1. **Algorithm for generating single-chain set.**

Input: state-chain

Output: state-set

1. Initialize an empty single-chain set in which there is currently only one empty single-chain, called the current chain l; Define that the position of the currently added element in l is k, and the initial value of k is 1. s is the state, and its initial value is the first element of the current recorded state-chain, that is, the state the agent currently reaches.

2. From the first element to the last element in the state chain, determine whether element s already exists in the single-chain set:

(1) if element s does not exist anywhere in the single-chain set, then add the element s to the position labeled k in the current chain, k plus 1;

(2) if element s already exists in the single-chain set, then define l^m as the single-chain with the smallest subscript of s in the state-set (since the single-chain in the state-set is also stored in reverse order, the smallest subscript of s means that s is the closest to the current state of the agent). The currently judged element $s = s_7$, and there are the following two single-chain containing s_7 in the current state-set:

$s_1 \leftarrow s_2 \leftarrow s_4 \leftarrow s_5 \leftarrow s_7 \leftarrow s_8$

$s_1 \leftarrow s_2 \leftarrow s_6 \leftarrow s_7 \leftarrow s_5$

Then the subscript of s is 5 in the first chain and 4 in the second chain. It can be seen that s_7 is closer to the current position s_1 in the second chain, and l^m should be the second chain.

Then *index* is defined as the subscript position of the state s in l^m. In this example, $index = 4$. Depending on the size of k and *index*, there are three cases:

i) $k > index$

The current chain l is $s_1 \leftarrow s_3 \leftarrow s_2 \leftarrow s_4 \leftarrow s_5 \leftarrow (s_7 k)$

The subscript of s to be added in the current chain is 6, which is larger than the subscript 4 of s in single-chain, that is, when moving from the state s to the first element s_1 in the state chain (i.e., the current position of the agent), the path moving according to the single-chain l^m is shorter than the path moving according to the current chain, which means the path stored in the single-chain l^m from s to s_1 is better, so the segment of the current chain from s to s_1 has no necessity to save.

So generate a new current chain l', move the elements from s_1 to s in l^m into the new current chain l', the old chain l only saves its path from the previous state s to the current state of the agent, and deletes the other states in l. The new single-chain can be depicted as follows:

$s_1 \leftarrow s_3 \leftarrow s_2 \leftarrow s_4 \leftarrow s_5$ (former current chain)
$s_1 \leftarrow s_2 \leftarrow s_6 \leftarrow s_7 \leftarrow (k)$ (new current chain)
$k \leftarrow$ length of current chain l' after change.

ii) $k = index$

This indicates that the path from s to the current state s_1 of the agent may have another solution with the shortest path, so the state s is directly added to the current chain, $k = k + 1$.

iii) $k < index$

This situation shows that there is a path shorter than any single-chain stored in the state-set from s to s_1, as follows:

$s_1 \leftarrow s_2 \leftarrow s_6 \leftarrow s_7 \leftarrow s_5(l^m)$
$s_1 \leftarrow s_3 \leftarrow (s_7k)$ (current chain)

Similar to the first case, the first single-chain l^m started from s_7 to s_1 is no longer necessary to be saved.

Create a new chain, copy the elements from s_1 to s_7 in the current chain into the new chain, and move the elements from the next element of s_7 to the last element in l^m into the new chain. The processing results can be depicted as follows:

$s_1 \leftarrow s_2 \leftarrow s_6(l^m)$
$s_1 \leftarrow s_3 \leftarrow s_7 \leftarrow s_5$ (new chain)
$s_1 \leftarrow s_3 \leftarrow s_7 \leftarrow (k)$ (current chain)

When all the states in the state chain are processed, the generated state set is the expected shortest loop free path set. Every time when the agent moves, the reward obtained will be successively transmitted to each single-chain according to the single-chain set, so as to update more Q values at a time.

3.2. **Q-learning algorithm based on single-chain set (set-Q-learning or SQ-learning).**

Place the agent in the initial state, after the agent moves to a state s', it first updates the corresponding Q-value according to the traditional Q-learning algorithm. If s' is the end state, it calculates the path state chain that the agent travels to form a single-chain set, and updates the Q-value according to the following algorithm.

Algorithm 3.1. SQ-learning algorithm

```
1: Input: state-set
2: Output: updated Q values
3: $i = 2$
4: when $i \leq$ the length of $l$, do:
    5: $prestate = l(i)$;
    6: $state = l(i - 1)$;
    7: $a^*$ = action taken from $prestate$ to $state$;
    8: $r$ = reward obtained by taking action $a^*$ at $state$;
    9: $\lambda(episodes)$ = learning rate;
    10: $Q(state, a^*) \leftarrow (1-\lambda)Q(state, a^*) + \lambda(r + \gamma^* \max_{a \in A} Q(prestate, a))$
11: $i = i + 1$
```

Through this algorithm, the agent can update more Q values by using local state knowledge in the process of exploration, which speeds up the learning speed.

3.3. **Algorithm of share single-chain library under multiple episodes (share-set-Q-learning or SSQ-learning).** If the single-chain set generated by the agent in multiple training episodes can be shared, each time the agent updates the Q value after reaching the end state, it can update more single-chains than that of the agent in a training episode without shared single-chain library, and the single-chains obtained will continue to increase with the increase of the training episodes. As a result, each time the agent reaches the end state, more Q value will be updated.

Definition 3.2. *Share-set is the single-chain library shared by the agent under different training episodes.*

The algorithm for generating share-set can be depicted as follows.

Algorithm 3.2. Algorithm for generating share-set

```
1: Input: state-set, the original share-set
2: Output: new share-set
3: When the agent reaches the end state, calculate the state-set according to the recorded state-chain;
4: For each single-chain in the state-set obtained in this run:
    5: Check whether the single-chain already appears in the share-set:
        6: If it already exists, skip the single-chain to the next cycle;
        7: If it does not exist, add the single-chain to the share-set.
8: Until each single-chain has been detected.
```

When the agent reaches the end state, the Q value is updated according to the share-set, so as to improve the update speed of the Q value.

3.4. **Calculating sub-targets based on the share-set.** According to the above algorithm, we have obtained the share-set of the agent under multiple training episodes, and the share-set contains the path situation experienced by the agent in each training episode. With the increase of the episodes, when path of the agent gradually tends to be stable, because all the single-chains stored in the share-set are non repetitive states, the probability that a state in the map is experienced by the agent can be calculated, so as to obtain the value of the state and which states in the map are more important can be determined, then sub-targets are set to make the path search of the agent more purposeful.

Assuming that the state value threshold that can be regarded as an important state is $value_{th}$, according to the state value calculation formula:

$$value = appear\text{-}times / total\text{-}line\text{-}num \tag{2}$$

The value of each state in the shared single-chain library can be known according to this formula, where *appear-times* denotes the occurrence times of a state in the share-set, and *total-line-num* is the total number of single-chains in the share-set. If the value of a state reaches the set threshold, it will be set as a sub target on the map. When the agent reaches this state, a positive reward will be given to the agent to encourage it to move to more important states. When the agent reaches a sub target, the reward is set as follows:

$$reward = basic\text{-}reward \Big/ \sqrt[10]{d} \tag{3}$$

where *basic-reward* represents a basic reward value, which is a constant, and d represents the distance from the sub target the agent reached to the end state. In this way, when the agent reaches the sub target closer to the end state, it can obtain greater reward, so as to guide the agent closer to the end state.

Because the shared single-chain library is dynamic, the sub targets are also dynamic. In the former several episodes, basically, all the actions of the agent are exploration actions. It is obviously inappropriate to calculate the sub target at this time. Therefore, the episode of the agent starting to calculate the sub target is set to 10, that is, the agent starts to look for the sub target after the 10th training episode. It is worth noting that to start calculating the sub target from the tenth training set is the best setting in our current research. Readers can try other settings according to their concrete researches.

4. SSQ-Learning with APF (Share-Set-Potential-Field Q-Learning or sspfQ-Learning).

In the traditional Q-learning algorithm, the agent does not have any understanding of the environment at the beginning of the environment exploration and selects actions completely randomly, so the convergence rate of Q value is slow. In order to make the agent have certain understanding of the environment when it begins to explore the environment and reduce the randomness of environment exploration, the APF method is introduced to generate the initial value of the Q table. In addition, this paper also combines the sub target with the APF algorithm to make the sub target produce a small attraction to the agent, and provides a certain guidance for the agent without destroying the gravitational potential field generated by the end state.

4.1. Algorithm introduction.

The Q table is initialized according to the start position of the agent, target position and obstacle position in the environment. The algorithm can be depicted as follows.

The APF generated by the target point can be set as

$$Q(s_x, \textit{all-action}) = C_{sc} + p_f * d^m(s_x, s_{end}) \tag{4}$$

where Q is the Q value corresponding to all actions in state s_x, C_{sc} is the gravity constant of the sub target, which can be set according to the map size, m is the APF factor, p_f is the gravitational gain factor, and $d(s_x, s_{end})$ represents the Euclidean distance from state s_x to s_{end}.

The repulsive potential field generated by the obstacle can be set as

$$Q(s_x, \textit{all-action}) = \begin{cases} p_r * \left(\dfrac{1}{d} - \dfrac{1}{d_0}\right) & d \le d_0 \\ 0 & d > d_0 \end{cases} \tag{5}$$

where p_r is the positive proportional gain factor of the repulsion potential field, d_0 is the maximal influence range of the repulsion potential field, beyond which the obstacle will no longer generate repulsion, and d is the distance from any position around the obstacle to the obstacle.

The final APF only needs to add the values generated by the gravitational potential field and the repulsive potential field. After initializing the Q table, the agent can have a certain understanding of the environment at the beginning of exploration, so as to accelerate the convergence.

4.2. APF combined with the sub target.

After obtaining the sub target, the exploration of the agent can be more purposeful by adding the APF to the sub target, so as to improve the learning speed. The APF of the sub target will only generate between the start point and the sub target point, so as to avoid unexpected influence on the APF generated by the final target point. The algorithm can be depicted as follows.

Algorithm 4.1. APF combined with sub target

```
1: Input: distribution of sub targets in the map
2: Output: updated Q table
3: If state $s$ is a sub target, do:
    4: For each state $s_x$ in the rectangle formed with the start state and sub target as vertex:
        5: Calculate the distance $d$ between $s_x$ and the sub target $s$;
        6: Calculate the distance $d_x$ from the new state $s_{x_{t+1}}$ after $s_x$ takes an action $a_x$ to the sub target $s$;
        7: If $d_x < d$, it indicates that taking action $a_x$ at the state $s_x$ can make the agent closer to the sub target, and update the Q value of the state-action pair with the following formula:
           $Q(s_x, a_x) = Q(s_x, a_x) + C_{sc} + p * d_m$   (6)
8: Until all sub target states have been experienced.
```

In Equation (6) d_m is the distance between the state s_x and the sub target; p represents the gravitational parameter of the sub target, and the calculation formula is set as follows:

$$p = -C_{sc}/D \tag{7}$$

where D is the distance from start state to the sub target. According to the formula, in the rectangular area formed by the sub target and the start state, the closer of the agent to the sub target after an action is taken in a certain state, the greater the positive reward can be obtained for the corresponding state-action pair. At the same time, because the positive update of Q value given by the algorithm only affects the state-action pair between the sub target and the start state, it will not cause the sub target to become a gravitational vortex and drag down other actions of the agent after passing through the sub target.

5. Improved Performance Based on the Path Number (Path-num-based-ϵ).

In the previous algorithm, we accelerate the learning of the agent by adding path backtracking and sub targets. Due to the random exploration of the agent, the final path of the agent may not be the optimal solution after Q-value convergence. To this end, the following improvements are made to the ϵ-*greedy* strategy used in the traditional Q-learning algorithm.

5.1. Improving the exploration rate.

In the above algorithms, the ϵ-*greedy* strategy is adopted and the exploration rate is set to $\epsilon = 1/episodes$. With the increase of the training episodes, the exploration rate will gradually drop to close to 0. Because this paper uses the SSQ-learning algorithm, compared with the traditional Q-learning, it needs more exploration to expand the share-set to make the calculation of sub targets more accurate. Therefore, this paper improves the exploration rate based on the number of the path solutions. In this algorithm, the exploration rate does not decrease inversely with the increase of training episodes, but increases or decreases according to the number of single-chains from the start state to target state extracted from shared single-chain library. Because the algorithm requires only the number of paths, only the single-chain from the initial state to the target state in share-set will be considered.

The algorithm dynamically adjusts the exploration rate of the agent, according to the number of different single-chains from the start point to the target point in the share-set, the specific algorithm can be depicted as follows.

Algorithm 5.1. Update the exploration rate based on the path solutions

```
1: Input: share-set
2: Output: updated exploration rate $\epsilon$
3: For each single-chain in the single-chain library, do:
4: If the single-chain is from the initial state to the target state, do:
    5: Check whether the same single-chain already exists in the solution set:
        6: If it does not exist, add the single-chain to the solution set, $result = result+1$;
        7: If it already exists, the total number of solutions remains unchanged;
8: If the single-chain is not a single-chain from the initial state to the target state, then directly detect the next single-chain;
9: Until all single-chains are traversed;
10: Update $\epsilon$ using the following formula:
```

$$\epsilon = \begin{cases} \epsilon + step_{renew} * (n_{\min} - n_{result}) & n_{result} < n_{\min} \\ \epsilon + step_{renew} * (episodes/f_e) & n_{result} \geq n_{\max} \end{cases} \tag{8}$$

where $step_{renew}$ is the updated step, $n_{\min}$ is the minimum required solution and $n_{\max}$ is the maximum required solution. These two constants can be set according to the complexity of the map; n_{result} is the number of recorded solutions, and f_e is the exploration factor, which is a constant, the larger it is, the smaller the value of ϵ reduced each time after reaching the highest requirement.

It can be seen from Formula (8) that the change of ϵ no longer only depends on the change of the number of training episodes, but also needs to consider the number of paths of the agent reaching the target state. When the number of paths is too few to meet the $n_{\min}$, it will increase the exploration rate and encourage the agent to explore more environment to obtain the optimal solution. Contrarily, when the number of solutions reaches the $n_{\max}$ and the number of solutions is sufficient, the exploration rate will reduce to obtain a higher convergence speed, which not only ensures the convergence speed, but also ensures that the agent is more likely to obtain the optimal solution.

5.2. **Improving the reward and discount rate.**

In the traditional Q-learning algorithm, the reward and discount rate γ are both set as constants, and the incentive for the agent to move towards the end point is not enough to make the agent converge to the optimal solution with high probability. Therefore, the reward and the discount rate γ are improved.

When the agent moves in the map and does not reach the target point, the improved reward and γ can be defined as follows:

$$reward = -d(s_{new}, a) * k_d \tag{9}$$
$$\gamma = C_\gamma - d(s_{new}, a) * k_\gamma \tag{10}$$

where $d(s_{new}, a)$ is the distance from the new state s_{new} to the target point after taking action a from a state s, k_d is the attenuation factor of the reward varying with the distance, C_γ is a constant, and k_γ is the attenuation factor of the γ varying with the distance. According to these two Formulas (9) and (10), the reward and γ obtained by the agent when taking different actions at different positions are dynamic, and the reward update encourages the agent to move closer to the target point. The update of γ makes the agent expect more future benefits when it is close to the target point, and reduces the impact of negative reward, so as to make the path of the agent tend to be the shortest and obtain the optimal solution.

6. **Promoting the Generalization Ability of Traditional Q-Learning Algorithm.**

In the former two parts of this paper, the slow convergence speed and low probability of convergence to the optimal solution of the traditional Q-learning algorithm are improved, but the insufficient generalization ability of the Q-learning algorithm has not been solved. This section adopts the improved Dyna-2 algorithm, which learns from the model, and combines the agent's experience in the real environment with the planning experience in the virtual environment, to make up for the generalization ability of the traditional Q-learning algorithm.

However, the path planning task in this work is carried out on the basis of the known environment model, that is, the agent is aware of both the state transition probability p (environmental dynamic characteristics) and the immediate reward R. Therefore, the virtual environment model no longer needs to be established by interaction with the real environment, and can be directly replaced by the real environment, and the interactive update in the virtual environment is no longer the temporary memory $Q'(s,a)$ but the permanent memory $Q(s,a)$. Therefore, the modified Dyna-2 can be depicted as follows:

Algorithm 6.1. Modified Dyna-2 algorithm

```
1: Input: episodes, $\alpha$, $\gamma$
2: Output: $Q$
3: Initialize the table of state-action value $Q(s,a)$
4: Circulation on each episode
    5: Initialization: initializes $s$ as the start state
    6: Iterations on each episode
        7: Search($s$): Interacts in the virtual environment with state $s$ and updates $Q$
        8: Action $a$ is obtained through the action selection policy
        9: Interact in the real world and get the reward $r$ and the next state $s'$
        10: $Q(s,a) = Q(s,a) + \alpha\left(r + \gamma \max Q(s',a') - Q(s,a)\right)$
        11: $s = s'$
        12: Until $s$ is the end state
13: Until the maximum number of circulations reaches
```

7. **Experimental Results and Analysis.**

In order to illustrate the effect of the improved Q-learning algorithm proposed in this paper, in this section, four simulation experiments are executed and compared their performance in two simulation environments.

7.1. **Parameter settings.**

The parameters of the traditional Q-learning algorithm are set as follows.

1) Setting of the reward:

$$r = \begin{cases} -1 & \text{Nomal movement} \\ -2 & \text{Collide with an obstacle} \\ 10 & \text{Reach the target} \end{cases} \tag{11}$$

2) Discount rate: $\gamma = 0.95$.

3) Exploration rate: $\epsilon = 1/episodes$.

4) Learning rate: $\alpha = 0.3$.

The parameters of the improved Q-learning algorithm are set as follows:

λ_a in Equation (1) is set to 0.35, $value_{th}$ is set to 0.9, *basic-reward* in Equation (3) is set to 1, and p_f, C_{sc} and m in Equation (4) are set to 1/30, 40 and 2, respectively. p_r and d_0 in Equation (5) are set to 6 and 3, respectively. $step_{renew}$, $n_{\min}$, $n_{\max}$ and f_e in

Equation (8) are set to 0.05, 10, 50 and 50, respectively. k_d in Equation (9) is set to 0.5, and C_γ and k_γ in Equation (10) are set to 0.98 and 1/30, respectively. The parameters are set and finally determined after many times trying. Namely, the performance of the current parameter setting is optimal for the path planning of the mobile robots in this work.

7.2. **Comparison of exploration strategy before and after modification.** In the first comparison experiment, the performance of different exploration strategies, i.e., the ϵ-greedy strategy and the improved exploration strategy based on the path number is first compared in the first simulation environment, as shown in Figure 1(a). The experiment results are provided in the Figure 2 and Table 1.

Figure 2 shows the number of attempts of the agent changing with its running times in the 50 runs under the two different action selection strategies. Further, the concrete data of the average, standard deviation, maximum and minimum of the number of attempts in the two action selection strategies are displayed in Table 1.

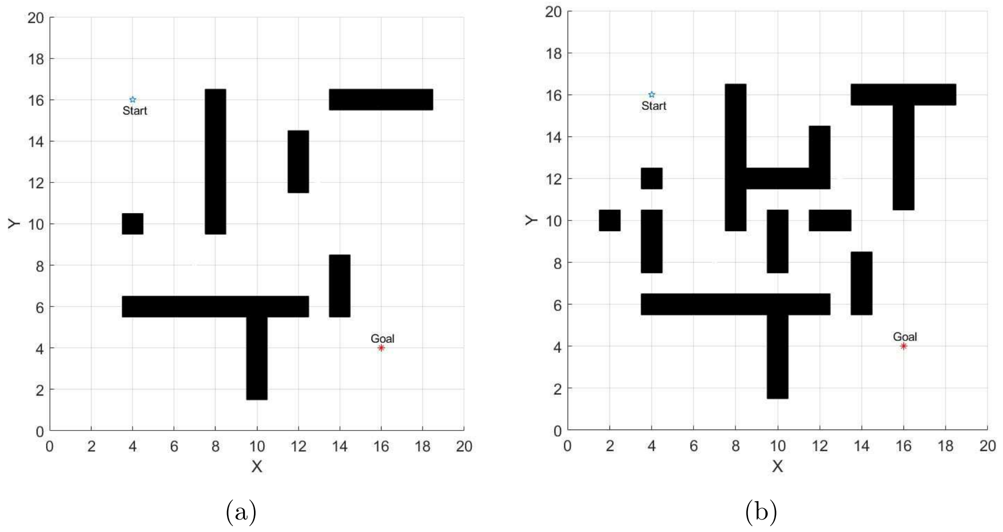

FIGURE 1. The first environment (a) and the second environment (b) used in the simulations

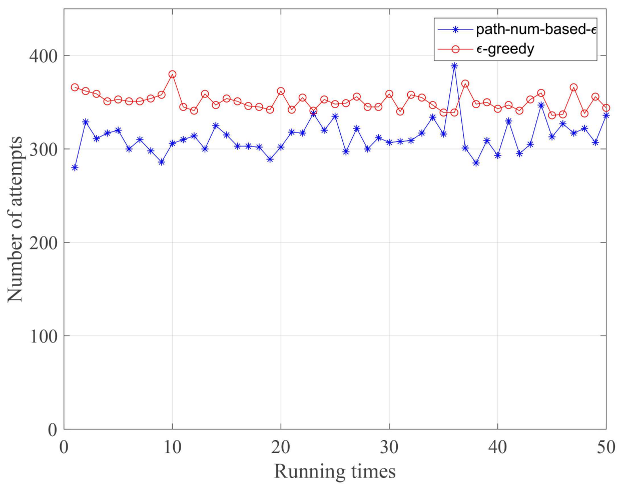

FIGURE 2. Number of attempts changing with the running times of the agent in the two different exploration strategies

TABLE 1. Performance comparison between the original ϵ-greedy strategy and the improved exploration strategy based on the path number

Algorithm	Average	Standard deviation	Max	Min
Path-num-based-ϵ	312.92	16.07	389	280
ϵ-greedy	350.74	10.22	380	336

In Table 1, the average, standard deviation, max and min denote the average, standard deviation, maximum and minimum of the training episodes when the algorithms converge. Table 1 shows that after the exploration strategy is updated, the convergence speed of the traditional Q-learning algorithm has been enhanced. That is, the average training episodes decreases form 350.74 to 312.92 in the 50 runs. At the same time, since the exploration rate based on the path number is adopted, when the agent takes the recorded action strategy for many times, the exploration rate may increase suddenly, resulting in unstable convergence speed. That is, the standard deviation of the training episodes when the algorithms converge increases from 10.22 to 16.07.

7.3. **Comparison between traditional Q-learning and sspfQ-learning.** In the second simulation experiment, the performances of the traditional Q-learning and the sspfQ-learning algorithms are compared and the experiment results are displayed in Figure 3 and Table 2.

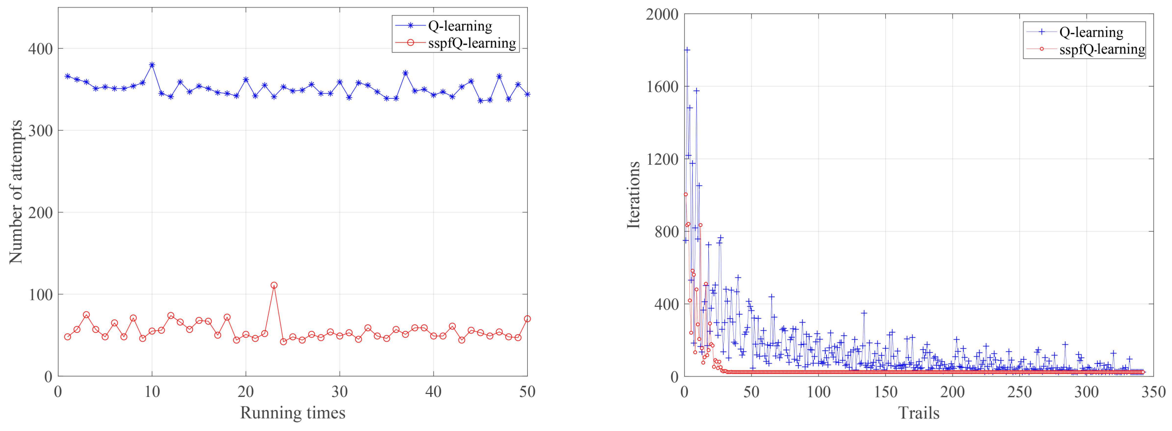

FIGURE 3. Number of attempts changing with the running times (a) and the iterations changing with the trial (b) between the traditional Q-learning algorithm and the sspfQ-learning

TABLE 2. Performance comparison between the traditional Q-learning algorithm and the sspfQ-learning

Algorithm	Average	Standard deviation	Max	Min
Q-learning	350.74	10.22	380	336
sspfQ-learning	55.54	11.78	111	42

Similarly, in Table 2, the average, standard deviation, max and min denote the same means as those in Table 1. From Table 2, we can see that after using sspfQ-learning, the convergence speed of agent with the sspfQ-learning algorithm has been greatly improved due to the use of chain updating method and the introduction of APF algorithm to speed up the learning speed of agent. That is, the average of the training episodes after the algorithm converges decreases from 350.74 to 55.54. The experimental results show that the upgraded Q-value updating method and the potential field introduced by APF can greatly accelerate the convergence speed of the Q-value. The reason is essentially to increase the number of Q values updated by the agent after completing an episode, and add a clearer goal orientation for the agent.

7.4. **Comparison between sspfQ-learning and Dyna-sspfQ-learning.** In the third experiment, the performances of the sspfQ-learning and Dyna-sspfQ-learning algorithms are compared and the corresponding results are displayed in Figure 4 and Table 3.

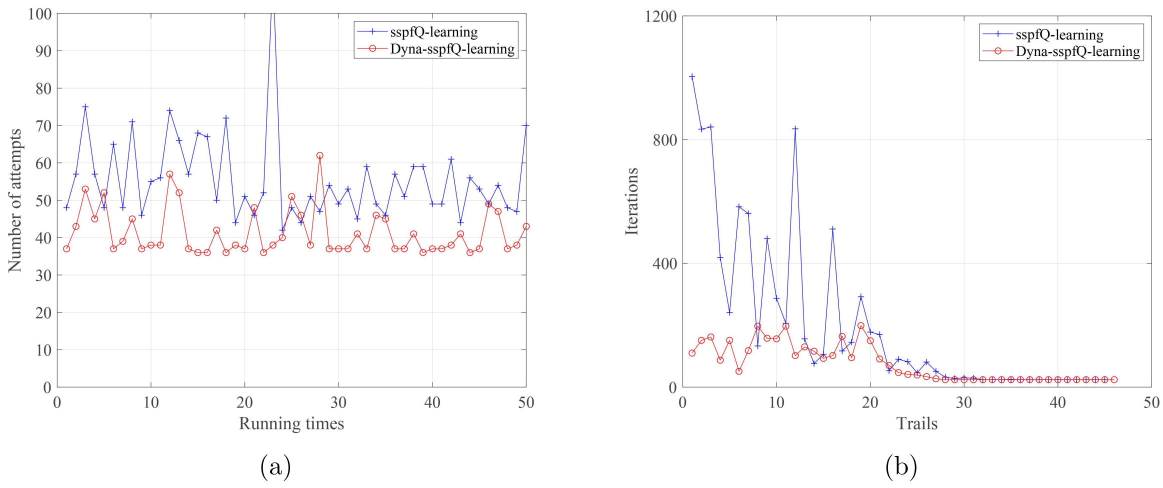

FIGURE 4. Number of attempts changing with the running times (a) the iterations changing with the trial (b) between the sspfQ-learning and the Dyna-sspfQ-learning

TABLE 3. Performance comparison between the sspfQ-learning and the Dyna-sspfQ-learning

Algorithm	Average	Standard deviation	Max	Min
sspfQ-learning	55.54	11.78	111	42
Dyna-sspfQ-learning	41.30	6.21	62	36

Similarly, in Table 3, the average, standard deviation (SD), max and min denote the same means as those in Table 1. From Table 3, we can see that after the introduction of virtual environment, the convergence speed of the Dyna-sspfQ-learning algorithm is improved because of the Q value updated in the virtual environment. Namely, the average of training episodes after the algorithm converges decreases from 55.54 to 41.30. At the same time, the convergence speed is more stable since the agent has explored and learned the map many times in the virtual environment and has a better understanding of the whole map. Namely, the standard deviation of the training episodes after the algorithm converges decreases from 11.78 to 6.21.

7.5. **Improvement of generalization ability with the improved Dyna-2 algorithm.** In the fourth experiment, the performance of the traditional Q-learning and the improved Dyna-2 algorithm are compared and the corresponding results are displayed in Figure 5 and Table 4.

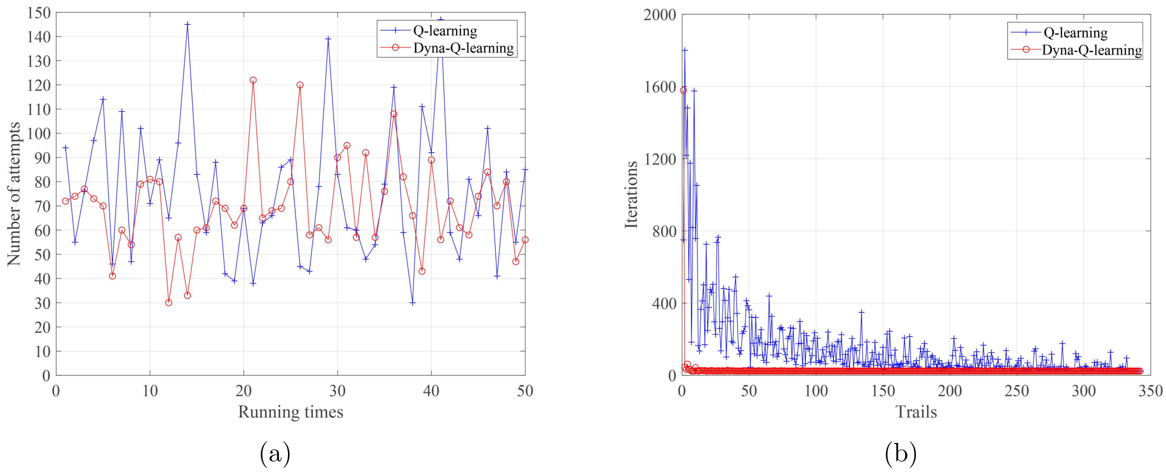

FIGURE 5. Number of attempts changing with the running times (a) the iterations changing with the trial (b) between the Q-learning and the Dyna-Q-learning

TABLE 4. Performance comparison between the Q-learning and the Dyna-Q-learning

Algorithm	Average	Standard deviation	Max	Min
Q-learning	75.94	28.20	147	30
Dyna-Q-learning	69.72	18.49	122	30

During the experiment, firstly, the traditional Q-learning and the improved Dyna-2 algorithms are used to plan the path in the simulation environment 1, as shown in Figure 1(a). After the two algorithms converge, the simulation environment is changed to scene 2, as shown in Figure 1(b), and the path planning of the above two algorithms is continued to verify the generalization ability of the improved Dyna-2 algorithm.

The results of the two algorithms in simulation environment 1 are displayed in Figure 5(a). Using the Q table obtained in first simulation scene, two algorithms are used to train in the simulation environment 2, and the corresponding results are shown in Figure 5(b).

Similarly, the corresponding experiment results are calculated and displayed in Table 4.

From Table 4, we can see that after using the improved Dyna-2 algorithm, the average of training episodes of the agent reduces from 75.94 to 69.72, and the corresponding standard deviation is also reduced from 28.20 to 18.49, indicating that after using Dyna-2 algorithm, the adaptability of agent to maps with high similarity is higher than that of traditional Q-learning algorithm, which shows that Dyna-2 algorithm improves the generalization ability of the Q-learning algorithm.

8. **Conclusion.** Aiming at the problems of slow convergence rate, difficulty in converging to the optimal solution and poor generalization ability of the traditional Q-learning algorithm, a novel improved Dyna-sspfQ-learning algorithm is proposed in this paper. In view of the slow convergence speed, this paper improves the discount rate, learning rate and other parameters of the traditional Q-learning algorithm, so as to improve the accuracy of value update. Meanwhile, a single-chain backtracking algorithm is introduced, which greatly improves the learning speed of the agent. To solve the invalid state loop in single-chain recorded by single-chain backtracking algorithm, an improved SSQ-learning algorithm based on single-chain set is designed to share the single-chain library under multi training episodes, by updating the Q value of the saved state in the shared single-chain library, and thus the convergence speed of the SSQ-learning is greatly improved. At the same time, in order to solve the random exploration of the agent in the early stage, the APF is introduced and combined with the sub targets obtained based on the shared single-chain library, to enhance the goal guidance of the agent. In order to solve the problem that the agent may converge to a non optimal solution due to the fast convergence speed in some cases, an improved exploration rate based on the path number is introduced. Finally, the improved Dyna-2 algorithm is designed to address the generalization capacity of the traditional Q-learning. Effectiveness of the improved Q-learning algorithm is verified by four comparative simulation experiments in two scenes.

Path planning of mobile robot in unknown environment is a complex problem. Although this paper puts forward some improvements, there are still many deficiencies to be further studied.

1) The algorithm in this paper still has some shortcomings. For example, although the improvement of exploration strategy increased the convergence speed, it also increased the instability of the training times in each episode. In addition, the adoption of single-chain set algorithm may lead to the excessive computation burden under complex maps.

2) This paper only carries out the simulation experiment of the improved algorithm, without applying the algorithm to the mobile robot. Next, experiments should be carried out on mobile robots to further demonstrate the potential of the proposed algorithm.

3) In the path planning problem in unknown environment, this paper only discusses the static environment, while the path planning problem in dynamic environment and irregular obstacles still needs further research.

Acknowledgment. This research is supported by the National Natural Science Foundation of China under Grant 62173309, Natural Science Foundation of Henan Province under Grant 202300410483, and Scientific Problem Tackling of Henan Province under Grant 192102210256.

REFERENCES

[1] X. Zhong, J. Tian, H. Hu and X. Peng, Hybrid path planning based on safe A* algorithm and adaptive window approach for mobile robot in large-scale dynamic environment, *Journal of Intelligent & Robotic System*, vol.99, pp.65-77, 2020.

[2] Y. Shin and E. Kim, Hybrid path planning using positioning risk and artificial potential fields, *Aerospace Science and Technology*, vol.112, no.6, 116640, 2021.

[3] Y. Zheng, J. Wang, D. Guo, H. Zhang, C. Li, D. Li, H. Li and K. Li, Study of multi-objective path planning method for vehicles, *Environmental Science and Pollution Research*, vol.27, pp.3257-3270, 2020.

[4] S. Pradhan, R. K. Mandava and P. R. Vundavilli, Development of path planning algorithm for biped robot using combined multi-point RRT and visibility graph, *International Journal of Information Technology*, vol.13, pp.1513-1519, 2021.

[5] J. Kober and J. Peter, Reinforcement learning in robotics: A survey, *International Journal of Robotics Research*, vol.32, no.11, pp.1238-1274, 2013.

[6] A. S. Polydoros and L. Nalpantidis, Survey of model-based reinforcement learning: Applications on robotics, *Journal of Intelligent & Robotic Systems*, vol.86, pp.1-21, 2017.
[7] J. Raajan, P. V. Srihari, J. P. Satya, B. Bhikkaji and R. Pasumarthy, Real time path planning of robot using deep reinforcement learning, *IFAC-PapersOnLine*, vol.53, no.15, pp.602-615, 2020.
[8] R. Watanuki, T. Horiuchi and T. Aodai, Vision-based behavior acquisition by deep reinforcement learning in multi-robot environment, *ICIC Express Letters, Part B: Applications*, vol.11, no.3, pp.237-244, 2020.
[9] J. Ou, X. Guo, M. Zhu and W. Lou, Autonomous quadrotor obstacle avoidance based on dueling double deep recurrent Q-learning with monocular vision, *Neurocomputing*, vol.441, pp.300-310, 2021.
[10] L. Li, D. Wu, Y. Huang and Z. Yuan, A path planning strategy unified with a colregs collision avoidance function based on deep reinforcement learning and artificial potential field, *Applied Ocean Research*, vol.113, 102759, 2021.
[11] G. P. Kontoudis and K. G. Vamvoudakis, Kinodynamic motion planning with continuous-time Q-learning: An online, model-free, and safe navigation framework, *IEEE Transactions on Neural Networks and Learning Systems*, vol.30, no.12, pp.3803-3817, 2019.
[12] X. Jin, Q. Jia, D. Ren and Y. Ba, Motion planning at intersection with event-driven recurrent Q-learning, *IFAC PapersOnLine*, vol.53, no.2, pp.17047-17052, 2020.
[13] L. Jiang, H. Huang and Z. Ding, Path planning for intelligent robots based on deep Q-learning with experience replay and heuristic knowledge, *IEEE/CAA Journal of Automatic Sinica*, vol.7, no.4, pp.1179-1189, 2020.
[14] L. Chang, L. Shan, C. Jiang and Y. Dai, Reinforcement based mobile robot path planning with improved dynamic window approach in unknown environment, *Autonomous Robots*, vol.45, pp.51-76, 2021.
[15] X. Liu, D. Zhang, T. Zhang, Y. Cui, L. Chen and S. Liu, Novel best path selection approach based on hybrid improved A* algorithm and reinforcement learning, *Applied Intelligence*, https://doi.org/10.1007/s10489-021-02303-8, 2021.
[16] S. P. H. P. Tabrizi, A. Reza and S. M. Jameii, Enhanced path planning for automated nanites drug delivery based on reinforcement learning and polymorphic improved ant colony optimization, *The Journal of Supercomputing*, vol.77, pp.6714-6733, 2021.
[17] Q. Yao, Z. Zheng, L. Qi, H. Yuan, X. Guo, M. Zhao, Z. Liu and T. Yang, Path planning method with improved artificial potential fields reinforcement learning perspective, *IEEE Access*, vol.8, pp.135513-135523, 2020.
[18] M. Zhao, H. Lu, S. Yang and F. Guo, The experience-memory Q-learning algorithm for robot path planning in unknown environment, *IEEE Access*, vol.8, pp.47824-47844, 2020.

Author Biography

Chaorui Chen is an undergraduate in Department of Automation, School of Electrical Engineering, Zhengzhou University, China. His research domain includes machine learning, bio-inspired computation, AI, etc.

Dongshu Wang received his Bachelor's degree in Mechanical Manufacture Technique and Equipment in 1996, Master's degree in Mechanical Manufacture and Automation in 2002, and Ph.D. in Control Theory and Control Engineering in 2006 from Northeastern University, China. His research domain includes autonomous mental development, artificial intelligence, machine learning and bio-inspired computation. Currently, he is a full Professor in School of Electrical Engineering, Zhengzhou University, Zhengzhou, China.