



A Dynamic Parameter Adaptive Path Planning Algorithm

Guangyu Yao, Nan Zhang^(✉), Zhenhua Duan, and Cong Tian

Institute of Computing Theory and Technology, and ISN Laboratory,
Xidian University, Xi'an 710071, China
nanzhang@xidian.edu.cn, {zhhdian, ctian}@mail.xidian.edu.cn

Abstract. Path planning in complex environments has always been a focus of research for scholars both domestically and internationally. This study addresses the challenge of path planning that combines obstacle avoidance and optimal path searching in scenarios lacking prior knowledge. The proposed approach introduces a parameter dynamic adaptation strategy for path planning. Experimental investigations are conducted using grid-based maps, and the results demonstrate that the method presented in this paper surpasses Q-learning and Sarsa algorithms in terms of comprehensive exploration, enhanced stability, and quicker convergence speed.

Keywords: Path planning · Reinforcement learning · ϵ -greedy policy

1 Introduction

Path planning is essential in real-world production, optimizing processes, increasing efficiency, cutting costs, and enhancing safety and flexibility. Optimal path planning and effective obstacle avoidance are critical challenges [7]. The objective is for agents to achieve the task of searching for a relatively optimal route from the starting point to the destination, utilizing excellent performance metrics [6, 13].

Path planning methods have a long history, and different algorithms yield varying results under different constraints. The traditional Dijkstra's algorithm [3] was introduced in 1956. This method was developed to address the single-source shortest path problem. The A* algorithm [4] is a path planning algorithm based on heuristic search. It combines cost and heuristic information, making it quite efficient. Genetic algorithms [8] can also be employed to solve path planning problems. Global search algorithms can find relatively optimal solutions in path planning but may have slower convergence speeds.

This research is supported by National Natural Science Foundation of China under Grant Nos. 62272359 and 62172322; Natural Science Basic Research Program of Shaanxi Province under Grant Nos. 2023JC-XJ-13 and 2022JM-367.

© The Author(s), under exclusive license to Springer Nature Switzerland AG 2024
W. Wu and J. Guo (Eds.): COCOA 2023, LNCS 14462, pp. 227–239, 2024.
https://doi.org/10.1007/978-3-031-49614-1_17

In many real-world applications, intelligent agents face increased uncertainty in their environments. This has led to the rise of reinforcement learning, gaining attention among researchers to address these challenges [12]. Reinforcement Learning [9, 10] is fundamentally a machine learning approach that involves learning through a “trial-and-error” process. In [11], neural networks are combined with the Q-Learning algorithm from reinforcement learning to address path planning problems in diverse environments. [1] introduces a novel approach by combining greedy and Boltzmann probability selection strategies as a means to avoid getting trapped in local optima. In [5], heuristic knowledge is integrated into reinforcement learning, resulting in improved efficiency for path planning and obstacle avoidance within the Deep Q-learning Network (DQN) algorithm. In [2], by optimizing convolutional neural networks in a specific way, the DQN algorithm is successfully employed to navigate through 3D mazes using visual information.

The main focus of this study is the application of reinforcement learning methods in path planning issue. The primary objective of this research is to enable intelligent agents to navigate even in unknown and relatively intricate scenarios. We propose a path planning strategy that involves dynamic parameter adaptation. The proposed strategy demonstrates more comprehensive search, better convergence.

2 Two Reinforcement Learning Algorithms

Reinforcement Learning is a machine learning algorithm based on the interaction between an agent and its environment, aiming to maximize the cumulative long-term reward through a sequence of decisions. The fundamental principles are illustrated in Fig. 1.

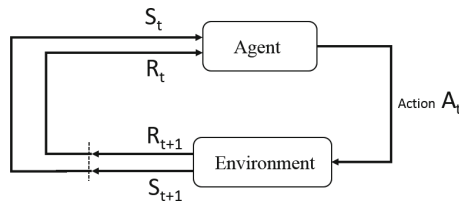


Fig. 1. Reinforcement Learning Diagram

Assuming the current step t , then S_t , S_{t+1} , A_t , R_t and R_{t+1} respectively represent the current state, the next state, the action, the current reward, and the next reward. The principle is to select the best action based on the current state and current reward, and generate the next reward and transition to the next state. Through learning an appropriate policy, the intelligent agent can make optimal action choices when facing different environmental states, with the aim of maximizing its expected cumulative reward.

2.1 Q-Learning Algorithm

The Q-learning algorithm is one of the most widely used techniques in reinforcement learning. Q represents the value associated with a state-action pair, representing the expected return when the agent takes action $a (a \in A)$ in state $s (s \in S)$. The environment provides an immediate reward r based on the agent's action a . Through continuous interaction between the agent and the environment, the algorithm updates the expected values based on the actual reward r obtained by taking action r in state s .

Q-learning employs a temporal difference (TD) method to update the value $Q_{(s_t, a_t)}$, and the update formula is as follows: $Q_{(s_t, a_t)} \leftarrow Q_{(s_t, a_t)} + \alpha(R + \gamma \max_{a'} Q_{(s_{t+1}, a')} - Q_{(s_t, a_t)})$. a' represents the action that yields the maximum value in the subsequent state. The pseudocode for the Q-learning algorithm is presented in Table 1.

Table 1. Q-learning Algorithm Pseudocode

Algorithm 2.1: Q-learning Algorithm
<p>Initialization: set the learning rate α; the discount factor γ; maximum number of iterations T; the value function $Q_{(s, a)}$</p> <p>while Repeat for each episode do Initialize the starting state s while Repeat for each step of the episode do Choose action a from state s Take action a, observe reward R and new state s' Update $Q_{(s, a)}$ using the Q-learning update rule: $Q_{(s, a)} \leftarrow Q_{(s, a)} + \alpha(R + \gamma \max_a Q_{(s', a)} - Q_{(s, a)})$ Update state $s \leftarrow s'$ end while end while</p>

2.2 Sarsa Algorithm

The distinction between the Sarsa algorithm and the Q-learning algorithm lies in the different strategies for selecting the next action and the timing of updating the value function $Q_{(s, a)}$.

In Q-learning, the next action is determined using a greedy policy, selecting the action with the highest Q value. In contrast, Sarsa employs an ε -greedy policy. Additionally, the timing of $Q_{(s, a)}$ updates also varies between the two algorithms. In Q-learning, the update of $Q_{(s, a)}$ takes place immediately after

executing the current action and selecting the next action. In Sarsa, the update of $Q_{(s,a)}$ occurs after executing both the current action and the selected next action.

The pseudocode for the Sarsa algorithm is presented in Table 2.

Table 2. Sarsa Algorithm Pseudocode

Algorithm 2.2: Sarsa Algorithm
<p>Initialization:</p> <p>set the learning rate α; the discount factor γ; maximum number of iterations T; the value function $Q_{(s,a)}$; The value of ε in the ε-greedy policy</p> <p>while Repeat for each episode do</p> <p style="padding-left: 2em;">Initialize the starting state s</p> <p style="padding-left: 2em;">while Repeat for each step of the episode do</p> <p style="padding-left: 4em;">Choose action a from state s</p> <p style="padding-left: 4em;">Take action a, observe reward R and new state s'</p> <p style="padding-left: 4em;">Choose next action a' from new state s'</p> <p style="padding-left: 4em;">Update $Q_{(s,a)}$ using the Sarsa update rule:</p> <p style="padding-left: 4em;">$Q_{(s,a)} \leftarrow Q_{(s,a)} + \alpha(R + \gamma Q_{(s',a')} - Q_{(s,a)})$</p> <p style="padding-left: 4em;">Update state $s \leftarrow s'$ action $a \leftarrow a'$</p> <p style="padding-left: 2em;">end while</p> <p>end while</p>

3 DPARL Algorithm

The Dynamic Parameter Adaptive-Reinforcement Learning (DPARL) algorithm is a reinforcement learning approach with dynamically changing parameters. Specifically, it involves the dynamic adjustment of the ε -greedy policy in the Sarsa algorithm. Depending on the progression of the path search process, whether at different episodes or different time steps within the same episode, the probability of random exploration varies accordingly. This adaptive approach is aimed at enhancing the interaction efficiency between the agent and the environment, ultimately improving the performance of the original algorithm.

Consider the path planning problem on a grid map as an example. During the initial exploration phase, the agent lacks a comprehensive understanding of the environment, necessitating substantial random actions for exploration. Towards the end of exploration, the agent should become more targeted in its exploration, requiring adaptive adjustments to its perception range. In the original strategy, the value of ε remains fixed. Thus, altering the ε factor in stages to accommodate the needs of path planning is crucial. The aim is to have a higher-than-average

likelihood of random exploration in the initial stage of the program and a lower-than-average likelihood towards the end.

The expression for the sigmoid function is as follows:

$$Sigmoid(x) = \frac{1}{1 + e^{-x}} \tag{1}$$

The graph of this function is depicted in Fig. 2. The graph illustrates that as x approaches negative infinity, the function value approaches 0 with a small slope. Conversely, as x tends towards positive infinity, the function value approaches 1 with a small slope. The function is bounded with values between 0 and 1. And the curve is smooth, continuously differentiable everywhere. Hence, the sigmoid function aligns with the desired probability reduction approach.

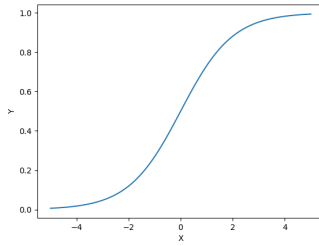


Fig. 2. Sigmoid Function

This paper proposes a path planning strategy called DPARL that improves Sarsa algorithm.

The ε -greedy policy is a commonly used trade-off strategy for exploration and exploitation in reinforcement learning. In the ε -greedy policy, ε is a positive number less than 1, typically between 0 and 1. The main idea of the strategy is that, during each action selection, the action with the highest estimated value is chosen with a probability of $1-\varepsilon$, while random exploration is conducted with a probability of ε by selecting a random action. Therefore, the value of ε controls the degree of exploration. For example, if $\varepsilon = 0.1$, then in each decision-making step, there is a 90% probability of selecting the action with the highest estimated value, and a 10% probability of selecting a random action. The parameter ε can be adjusted using the following formula:

$$\varepsilon = \frac{\rho}{1 + e^{-\left(\frac{\max_iterations}{2} - n_iter\right)}} + \phi \tag{2}$$

In the initial stage of the program, when $n_iter - \frac{\max_iterations}{2}$ is at its minimum, $\frac{\max_iterations}{2} - n_iter$ is at its maximum, $\varepsilon \approx \rho + \phi$ is at its maximum. As the program progresses, when $n_iter - \frac{\max_iterations}{2} = 0$, $\frac{\max_iterations}{2} - n_iter = 0$, $\varepsilon = \frac{\rho}{2} + \phi$. In the final stage of the program, when $n_iter - \frac{\max_iterations}{2}$ is at its maximum, $\frac{\max_iterations}{2} - n_iter$ is at its minimum, $\varepsilon \approx \phi$ is at its minimum.

Hence, by appropriately setting the parameters ρ and ϕ according to the specific problem, the desired probability reduction approach will be satisfied.

For a single episode's iteration cycle, ε can also be dynamically adjusted. When the agent is farther from the target location, set ε to a value higher than the average probability, promoting exploration. Conversely, when the agent is closer to the target location, set ε to a value lower than the average probability, focusing on exploitation. Using μ to represent the adaptive factor for the fluctuation of ε . As shown in the formula 3 (ε in formula 3 is the same as ε in formula 2):

$$\varepsilon_{episode} = \frac{\mu}{1 + e^{-[(row_current + col_current) - \frac{row + col}{2}]}} + \varepsilon - \frac{\mu}{2} \quad (3)$$

In a single episode iteration, when the agent is at the farthest distance from the target position, $(row_current + col_current) - \frac{row + col}{2}$ is at its maximum, $\varepsilon_{episode} = \varepsilon + \frac{\mu}{2}$ is at its maximum. This encourages exploration with a probability higher than the average ε . When the agent is closest to the target position, $(row_current + col_current) - \frac{row + col}{2}$ is at its minimum, $\varepsilon_{episode} = \varepsilon - \frac{\mu}{2}$ is at its minimum. This leads to exploration with a probability lower than the average ε .

The parameter dynamic adjustment path planning strategy adapts the random exploration probability, enhancing the efficiency of exploring the environment. This strategy better balances the exploration-exploitation trade-off, improves the interaction efficiency between the agent and the environment, and

Table 3. DPARL Algorithm Pseudocode

Algorithm 3.1: DPARL Algorithm

Initialization:

set the learning rate α ; the discount factor γ ; maximum number of iterations T ; the adaptive factor ρ ; the adaptive factor ϕ ; the adaptive factor μ ; the value function $Q_{(s,a)}$; the value of ε in the ε -greedy policy

while Repeat for each episode **do**

 Initialize the starting state s

 Update the value of ε

while Repeat for each step of the episode **do**

 Update the value of $\varepsilon_{episode}$

 Choose action a from state s through ε -greedy policy

 Take action a , observe reward R and new state s'

 Update $Q_{(s,a)}$ using the Sarsa update rule:

$$Q_{(s,a)} \leftarrow Q_{(s,a)} + \alpha(R + \gamma Q_{(s',a')} - Q_{(s,a)})$$

 Update state $s \leftarrow s'$ action $a \leftarrow a'$

end while

end while

enhances the performance of the original algorithm. The pseudocode for the DPARTL algorithm is presented in Table 3.

4 Experimental Setup and Experimental Results

4.1 Description of the Path Planning Problem

The experimental environment consists of maps constructed using grid cells, as depicted in Fig. 3. There are two maps: one composed of 5×5 grid cells and the other of 10×10 grid cells. Gray grid cells represent obstacles, white grid cells depict possible paths, red dots indicate the starting points, and blue dots mark the destination points. The agent begins from the starting coordinates and navigates to the destination point while passing only through white grid cells.

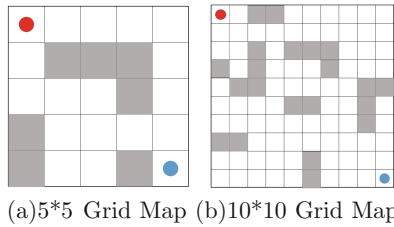


Fig. 3. Grid Maps (Color figure online)

4.2 Modeling and Parameter Settings for the Path Planning Problem

4.2.1 State and Action Set

The starting grid cell serves as the initial state, the termination grid cell denotes the goal state, while the collection of all white grid cells constitutes the state set. The set of actions comprises valid actions after excluding illegal ones, defined as actions that would lead the agent to step outside the grid boundaries. The set of actions can be represented using coordinate transformations.

4.2.2 Q-Matrix

The matrix contains probabilities corresponding to state-action pairs.

4.2.3 Reward Function

When the agent moves out of the map boundaries or enters a gray grid cell, it receives a punitive reward of -10 . When the agent reaches the target location, it receives a reward of 10 . As the agent gets closer to the target location, it receives a reward of 0 . If the agent moves away from the target location, it receives a punitive reward of -0.2 .

4.2.4 Partial Parameter Settings

For a 5*5 map, the iteration number T is set to 100. For a 10*10 map, the iteration number T is set to 500. Learning rate α is 0.1. Discount factor γ is 0.9. Greedy policy parameter ε is 0.2. Dynamic adaptive factor ϕ is 0.15. ε adjustment factor μ is 0.05.

4.2.5 Experimental Environment

The experiments were conducted on a machine with 64 GB of RAM, using VMware Workstation 15 as the virtualization software. The operating system employed was Ubuntu 20.04, with 16 GB of virtual memory allocated. The programming language used for the experiments was Python 3.

4.2.6 Evaluation Metrics

- (1) Runtime: The time taken for the algorithm to perform one path planning task. Shorter runtime indicates lower resource consumption of the algorithm.
- (2) Average steps: The average length of paths generated by the algorithm in one path planning task.
- (3) First detection of optimal path time: The time taken by the algorithm to first detect the expected optimal path in one path planning task.
- (4) First detection of optimal path iteration number: The iteration number at which the algorithm first detects the expected optimal path in one path planning task.

4.3 Experimental Results and Analysis

The experiment designed two scenarios: a 5*5 grid map and a 10*10 grid map. Path planning was conducted using three different methods in each scenario. Optimal path diagrams were generated and the performance of the three methods was compared.

4.3.1 5*5 Grid Map Scenario

In Experiment Scenario 1, it's evident that there is more than one shortest path. Even among paths with the same shortest length, some algorithms might favor turns while others might prefer straight-line paths. In this experiment, each algorithm was trained 100 times, and the resulting path data was collected. The path that appeared most frequently was chosen as the optimal path for that algorithm. The optimal paths generated by these three algorithms in this experiment are displayed in Fig. 4, where the red dashed line represents the optimal path for each algorithm.

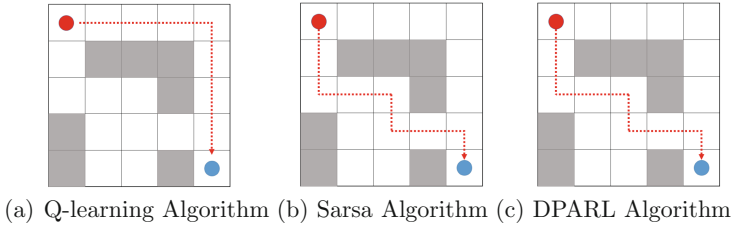


Fig. 4. Three Different Algorithms’ Optimal Paths (Color figure online)

From Fig. 4, it is apparent that all three algorithms can provide optimal paths. However, the Q-learning algorithm exhibits fewer turns, while the other two algorithms display more frequent turns, indicating that the Q-learning algorithm tends to favor straight-line exploration and is less likely to change its behavioral habits. Conversely, this suggests that the other two algorithms lean towards comprehensive exploration, resulting in a more evenly distributed behavior pattern. In addition to this, the paper also compares the performance of the three algorithms by calculating the average runtime, average step count, time taken for the first detection of the optimal path, and the number of iterations for the first detection of the optimal path over 100 runs. The results are summarized in Table 4.

Table 4. Performance Comparison of Three Algorithms (Averaged Over 100 Runs)

Algorithm	Q-learning	Sarsa	DPARL
Average Runtime (ms)	77.8	52.5	53.6
Average Step Count(steps)	17.6	14.6	13.5
Time for 1st Optimal Path(ms)	18.3	17.2	11.9
Iterations for 1st Path	12.5	6.2	4.6

From Table 4, it is evident that among the three algorithms, the Sarsa algorithm consumes the least amount of time and Q-learning requires the longest time. Looking at the average step count, both the DPARL and Sarsa algorithms exhibit similar step counts, both significantly lower than the Q-learning algorithm. Analyzing the time taken for the first detection of the optimal path, the DPARL algorithm demonstrates the shortest time and the Q-learning algorithm takes the longest time. In terms of the number of iterations required for the first optimal path, DPARL demands the fewest iterations, followed by Sarsa, and Q-learning requires the most iterations. This observation suggests that the DPARL algorithm is capable of finding the optimal path with fewer iterations and less time, showcasing excellent stability and a lower frequency of encountering invalid paths.

In Scenario 1, the path planning was performed using three different algorithms, and the reward curves for single training runs are depicted in Fig. 5. The horizontal axis represents the current iteration episodes, while the vertical axis represents the total reward obtained in each episode.

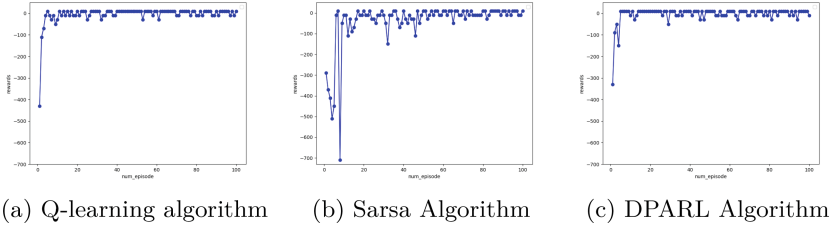


Fig. 5. The rewards curves for the three different algorithms

From Fig. 5, it can be observed that when using the DPARL algorithm, the agent is able to reach the optimal path earlier and more frequently. On the other hand, the Q-learning algorithm and Sarsa algorithm take longer to converge to the optimal path and exhibits more instability.

4.3.2 10*10 Grid Map Scenario

In Experiment Scenario 2, similar to Experiment Scenario 1, each algorithm was trained 100 times, and the most frequent path result among the trials was selected as the best path for that algorithm. The best paths obtained by the three algorithms in this experiment are illustrated in Fig. 6, where the red dashed lines represent the optimal paths determined by each algorithm.

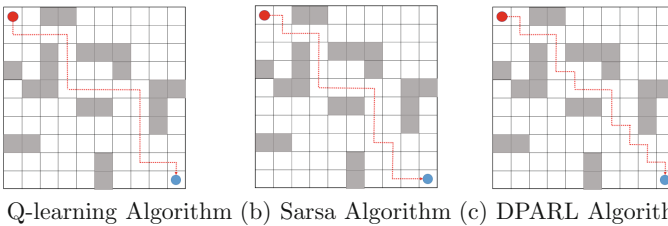


Fig. 6. Three Different Algorithms' Optimal Paths (Color figure online)

From Fig. 6, it is evident that all three algorithms are capable of providing optimal paths. However, Q-learning algorithm demonstrates fewer turns in comparison to Sarsa algorithm, while the DPARL algorithm exhibits more turns. In addition to this, the paper also compares the performance of the three algorithms

by calculating the average runtime, average step count, time taken for the first detection of the optimal path, and the number of iterations for the first detection of the optimal path over 100 runs. The results are summarized in Table 5.

Table 5. Performance Comparison of Three Algorithms (Averaged Over 100 Runs)

Algorithm	Q-learning	Sarsa	DPARL
Average Runtime (ms)	7707.2	517.5	490.7
Average Step Count(steps)	597.4	36.9	32.9
Time for 1st Optimal Path(ms)	7299.1	201.3	161.5
Iterations for 1st Path	135.4	80.3	66.5

From Table 5, it is evident that among the three algorithms, DPARL algorithm has the shortest runtime, while Q-learning’s runtime is 14 times that of the former two. Looking at the average number of steps, both DPARL and Sarsa algorithms exhibit similar values, significantly fewer than Q-learning algorithm. Regarding the time taken for the first detection of the optimal path, DPARL algorithm performs the best and Q-learning has the longest time. In terms of the number of iterations required for the first detection of the optimal path, DPARL algorithm outperforms the others, while Q-learning requires the most iterations. This indicates that DPARL algorithm is capable of finding the optimal path with fewer iterations and in less time, demonstrating good stability, and resulting in fewer occurrences of ineffective paths.

In Scene 2, the path planning was performed using three different algorithms. The reward curves for each algorithm in a single training run are shown in Fig. 7. The horizontal axis represents the current iteration episodes, while the vertical axis represents the total reward obtained in each episode.

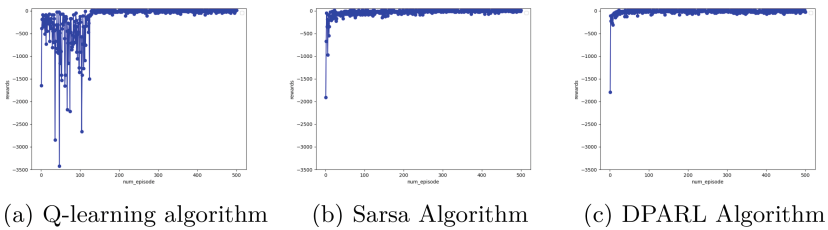


Fig. 7. The rewards curves for the three different algorithms

From Fig. 7, it is evident that when using the DPARL algorithm, the intelligent agent is able to reach the optimal path earlier and more frequently. As the complexity of the scene increases, the DPARL algorithm exhibits strong stability, avoiding the occurrence of overly long paths. This algorithm appears to be well-suited for path planning in intricate maps.

5 Conclusions

In complex environments lacking prior environmental information, traditional path planning algorithms suffer from drawbacks such as high computational complexity, low efficiency, and unstable results. To address this challenge, this paper proposes a path planning strategy called DPARK. This strategy increases the random search probability in the early stage and decreases it in the later stage of the program, enhancing the interaction efficiency between the agent and the environment, and improving the performance of the existing algorithm. The paper compares the DPARK with the Q-learning algorithm and the Sarsa algorithm. The DPARK demonstrates more comprehensive search, better convergence, and the ability to find optimal paths with fewer iterations and less time. Additionally, it maintains high stability even in complex maps.

In future work, further enhancement of the adaptability of the adaptive algorithm is needed. Additionally, applying parameter adaptive path planning algorithm to high-precision modeling environments is also a potential research area.

References

1. Chen, H., Ji, Y., Niu, L.: Reinforcement learning path planning algorithm based on obstacle area expansion strategy. *Intell. Serv. Robot.* **13**(2), 289–297 (2020)
2. Devo, A., Costante, G., Valigi, P.: Deep reinforcement learning for instruction following visual navigation in 3D maze-like environments. *IEEE Rob. Autom. Lett.* **5**(2), 1175–1182 (2020)
3. Dijkstra, E.W.: A note on two problems in connexion with graphs. In: Edsger Wybe Dijkstra: His Life, Work, and Legacy, pp. 287–290 (2022)
4. Hart, P.E., Nilsson, N.J., Raphael, B.: A formal basis for the heuristic determination of minimum cost paths. *IEEE Trans. Syst. Sci. Cybern.* **4**(2), 100–107 (1968)
5. Jiang, L., Huang, H., Ding, Z.: Path planning for intelligent robots based on deep q-learning with experience replay and heuristic knowledge. *IEEE/CAA J. Automatica Sinica* **7**(4), 1179–1189 (2019)
6. Patle, B., Pandey, A., Parhi, D., Jagadeesh, A., et al.: A review: on path planning strategies for navigation of mobile robot. *Defence Technol.* **15**(4), 582–606 (2019)
7. Polydoros, A.S., Nalpantidis, L.: Survey of model-based reinforcement learning: applications on robotics. *J. Intell. Rob. Syst.* **86**(2), 153–173 (2017)
8. Santiago, R.M.C., De Ocampo, A.L., Ubando, A.T., Bandala, A.A., Dadios, E.P.: Path planning for mobile robots using genetic algorithm and probabilistic roadmap. In: 2017IEEE 9th International Conference on Humanoid, Nanotechnology, Information Technology, Communication and Control, Environment and Management (HNICEM), pp. 1–5. IEEE (2017)
9. Sutton, R.S., Barto, A.G.: Reinforcement Learning: An Introduction. MIT press, Cambridge (2018)
10. Szepesvári, C.: Algorithms for Reinforcement Learning. Springer, Heidelberg (2022). <https://doi.org/10.1007/978-3-031-01551-9>
11. Wei, J., De Hua, Z., Shuangbao, M., Gaocheng, Y., Wei, C.: Dynamic walking characteristics and control of four-wheel mobile robot on ultra-high voltage multi-split transmission line. *Trans. Inst. Meas. Control.* **44**(6), 1309–1322 (2022)

12. Yang, Y., Juntao, L., Lingling, P.: Multi-robot path planning based on a deep reinforcement learning DQN algorithm. *CAAI Trans. Intell. Technol.* **5**(3), 177–183 (2020)
13. Zhang, H.Y., Lin, W.M., Chen, A.X.: Path planning for the mobile robot: a review. *Symmetry* **10**(10), 450 (2018)