Path Planning with Q-Learning

Yuepeng Hu1,†,*, Lehan Yang2,†, Yizhu Lou3,†

1North Carolina State University, Raleigh, NC 27695, United States;
2Shattuck-St. Mary’s School, Faribault, MN 55021, United States;
3University of California Irvine, Irvine, CA 92697, United States.

*Corresponding author e-mail: yhu13@ncsu.edu
†These authors contribute equally.

Abstract. As science and technology advance rapidly, the scope of mobile robot applications continues to expand. In mobile robotics, path planning, which has formed a relatively complete theoretical system, is one of the most momentous and challenging tasks, especially in uncertain environments. Path planning has been a popular research problem in recent years, and it has been widely utilized in various fields, mainly including robotic surgery, logistics transportation, and self-driving automobile. To solve path planning, this paper presents an approach that uses Q-learning Algorithm to find multiple feasible paths within obstacle environments. First, the algorithm of Q-learning was pre-trained to make it suitable for path planning. Then an obstacle environment map was modeled and a path planning program was compiled by applying the state action-value function. Finally, the experimental results were collected, including the path planning maps and corresponding graphs of training progress. To guarantee the effectiveness of the final paths, all the parameters in the function are set to be constant. The experimental results show that the Q-learning algorithm can succeed in solving the problem of multi-path planning.

1. Introduction

Path planning is a computation problem that is an important and essential skill in the robotics field. The main purpose of path planning is to find the best route between the start and the endpoint. For most robots, the optimum path planning is the shortest path between two locations. However, some robotics need to plan a path that achieves some requirements to deal with uncertain conditions. For example, vacuum cleaning robots and automated lawnmowers need to find a path that covers all the spaces between the two points [1]. In this situation, implementing the shortest-route strategy is not a wise option.

To solve path planning, the A* algorithm, which is a powerful approach for path planning to search for the single shortest path, estimates the minimum distance between two locations to choose the optimum path [2]. However, according to Lee [3], the multi-path routing strategy has a more outstanding performance than the shortest-path navigating strategy in a vehicle route guidance system especially when the road network is congested. Specifically, if a number of vehicles navigate on the same generated path, the route will be overloaded. To avert the deterioration, a number of alternative paths are necessary to be provided for other adjacent vehicles on the network so that they can be spatially diffused. Therefore, providing numerous candidate paths is obviously a practical and effective approach.

On the other hand, A* is an informed search algorithm that extremely relies on calculating the movement cost from a given position to the destination [4]; that means if the position of the destination is not known, using A* algorithm will fail to find the shortest path even a feasible path. In reality, there
are usually some other real-world applications where complete mapping data is unachievable. One such example is self-driving: it is almost impossible for all roads to be mapped, and even then, any change in the roadways, such as weak signal, may cause the vehicle to navigate to the wrong place. Thus, a pre-mapped algorithm would not only endanger the driver but also other road users. Due to the problem of self-driving remaining challenging and unsolved, vehicles could not automatically drive without a driver currently. Q-learning [5], which is a reinforcement learning algorithm [6], can make an agent learn to discover the most valuable actions to achieve the goal in an uncertain environment, and it can produce multiple paths.

To overcome the two limitations of the A* algorithm, this paper presents to use Q-learning for path planning in a virtual obstacle environment to find multiple routes. An obstacle environment has been created first by generating a grid-based map. In the map, obstacles are in a distinct color from free space, and the origin and the destination are stationary. The state action-value function [7], which is the function of Q-learning, is applied to tell the current status and select actions based on the Q-values, and all the parameters in the function are constant. A Q-values Table is created through this learning process, and the training progress is recorded and graphed with the final path. When various feasible paths are generated, the experiment is considered to be successful. According to the experimental results and analysis, if Q-learning can seek multiple paths or not by according to the experimental results.

2. Method
This section introduces the detailed process of implementing Q-learning for path planning. A grid map is created as the virtual obstacle environment in which the path is to be planned. The final path planned is to be projected onto the map while an animated plot of training progress is shown in a separate figure. This entire section is divided into three subsections: Pre-Processing, which talks in detail about the creation of the environment, Path Planning with Q-learning Algorithm, which gives our detailed description on how Q-learning is implemented for path planning, and Plotting, which introduces how the result is represented.

2.1. Pre-processing
Before running any path planning algorithm, an environment is needed to exist for the algorithm to run in. In this paper, a virtual obstacle environment is created. To create an environment in MATLAB, an empty environment is initialized first. The environment (grid map) is represented by a table in which each cell represents a coordinate, thus filling it with “0” indicates that the environment is free of obstacles. Then, the endpoint could be added. For the endpoint to be differentiated from regular and obstacle coordinates, its value is set to be “1”. Finally, the obstacles are created whose values in the table are set to be “-1”. With such an environment created, the algorithm is implemented as the endpoint and obstacles could be recognized and used for training.

2.2. Path Planning with Q-learning Algorithm
In the experiment, the Q-learning Algorithm for path planning is utilized. “Q” defines how useful an action that an agent takes in a state is [8], and it is computed by using the state action-value function, which is as follow:

\[
\Delta Q_t(s_t, a_t) = \alpha [r_{t+1} + \gamma \max Q_t(s_{t+1}, a_{t+1}) - Q_t(s_t, a_t)]
\]

(1)

The Q-learning Algorithm utilizes a Q Table, and the Q Table is updated according to the \(\alpha\) and \(\gamma\) value. The Q Values in the Q Table are used to determine what the best action is.

The detailed process of Q-learning for path planning is discussed in this subsection.

2.2.1. Algorithm Parameters. The two main parameters in the Q-learning Algorithm are \(\alpha\) and \(\gamma\). \(\alpha\) is the learning rate [9], also known as the step size, which determines to what extent newly acquired information overrides old information. A low learning rate indicates that old information is preferred over new information and a high learning rate indicates the opposite. \(\gamma\) is the discount factor [9], which determines how much the future rewards are valued. A low discount factor indicates that the algorithm
prefers rewards in the short term, while a high discount factor would value rewards in the future more. The $\alpha$ and $\gamma$ value can be changed to favor either existing or new data, shorter or longer path.

2.2.2. Actions and Q Table Initialization. The environment is a grid map, and each 1 by 1 grid is defined to be one state. At each state, there are only four potential actions: up, down, left, and right. Since the Q Table includes the Q Values of every action at every state, the Q Table needs to be a three-dimensional array. Each layer of the Q Table is the same as the grid map, and each number in each layer of the Q Table corresponds to one state in the map. There are four layers in the Q Table, each layer corresponds to one of the four potential actions. Before running the algorithm, the Q Table must be created with a neutral value ("0") for all the numbers in the table.

2.2.3. Action Selection, Rewards, and Q Table Update. When running the algorithm, the agent uses the Q Table to select an action with the highest Q Value. When the Q Values of different actions are equivalent, such as at the beginning, the agent would select a random action. The Q Values are updated according to the algorithm parameters when the agent receives rewards. If the agent hits a boundary or an obstacle, a negative reward is given. On the other hand, if the agent arrives at the endpoint, the agent will receive a positive reward "+1". For all the other actions, the agent receives no reward. If the agent’s actions resulted in no rewards, the Q Values would remain the same as before. If the agent receives a reward, the Q-Table would be updated according to the Q-learning Algorithm.

2.3. Visualization.
To illustrate the experimental results, a final path is plotted on the environment map. The final path is plotted by choosing the action with the highest Q Value at every state from the start point $(1,1)$ to the endpoint. The plotted path offers a visual representation of the path that the agent would choose with the dataset (Q Table) at the end of the training period.

3. Experimental Results
This section shows the experimental settings and detailed experimental results to verify the proposed method for path planning.

3.1. Experimental Settings
For this experiment, there is an obstacle environment based on a 20 by 20 grid. In fig. 1, the black line around the outside shows the boundary, and the black line inside shows the grid. Green shows parts of the map that are free of obstacles, while blue represents the obstacles. In the obstacle environment, the starting point is at the bottom left $(1,1)$, and the endpoint is shown in yellow, at the top right $(20,20)$. The obstacle environment created includes four obstacles of different sizes, each in different locations on the map. In this environment, multiple possible paths are of similar length, making it a great environment to test whether Q-learning can find multiple paths.

Fig. 1. The created environmental map
In the experiment, $\alpha$ is set to 0.5, $\gamma$ is set to 0.1. 0.5 is a moderate value for the learning, which indicates that the algorithm would keep old information while also allowing new information to be acquired. The discount factor is set to 0.1 because this paper explores a path planning scenario which generally prefers shorter path to longer ones. A discount factor of 0.1 would indicate that shorter paths are rewarded for than longer ones.

The experiment result is effective if the endpoint set is reached. We can determine whether it was reached by observing the visual representation of the final planned path or whether a “+1” reward given in the Q Table.

3.2. Experimental Results

According to the experimental results, Q-learning can successfully find multiple paths from the start point to the endpoint without given map data. The three most representative runs are visualized, but each with the same settings. For each run, there is a corresponding figure of training progress and an action table. The first example is shown in Fig. 2.

![Fig. 2. The first planned path](image)

Q Table is created through the training progress in each run, and it is a three dimensional $(20 \times 20 \times 4)$ array that indicates the preferred action at each position. The four actions indicated in the table are as follow: action “1” is moving down, action “2” is moving right, action “3” is moving up, action “4” is moving left. For instance, Table. I. demonstrates the Q-values corresponding to the four actions at position $(1,1)$.

| Action 1 | Action 2 | Action 3 | Action 4 |
|----------|----------|----------|----------|
| -0.5000  | 0        | 0.0000   | -0.5000  |

According to the Table. I, action “3” has the biggest value, which is something bigger than 0.0000. Therefore, at the status $(1,1)$, an action of moving up should be taken, and it matches the initial action in Fig. 2.

Fig.3 illustrates the rewards of each selected action. The episodes are set to be 1000, and the initial reward is also recorded; thus, 1001 rewards should be shown below.
The two figures (Fig. 4 & Fig. 5) shown below are the other paths planned at two individual runs. As shown in these figures, Q-learning is able to find multiple paths that lead to the same end point without being given map data.

4. Conclusion
In this paper, a Q-learning algorithm is proposed to seek several collision-free paths in obstacle environments. An obstacle environment map and implement the logic of the Q-learning algorithm is created to plan a feasible path in the environment with multiple obstructions. Additionally, the rewards of each state through the whole episodes are recorded to make a training progress plot. The experimental results show that it is capable of using a cooperative Q-learning algorithm to execute path planning and find multiple feasible paths.

In the future, the study will focus on seeking an optimal path and suboptimal multipath by Q-learning algorithm in urban road networks. However, due to the problems of low efficiency and weak capacity of path optimization, which are the weaknesses of the Q-learning algorithm, it is necessary to consider overcoming these difficulties. Therefore, the problem of inefficiency will also be studied and solved.
References
[1] Piardi, Luis, et al. “Coverage path planning optimization based on Q-learning algorithm.” AIP Conference Proceedings. Vol. 2116. No. 1. AIP Publishing LLC, 2019.
[2] Delling, Daniel, et al. “Engineering route planning algorithms.” Algorithmics of large and complex networks. Springer, Berlin, Heidelberg, 2009. 117-139.
[3] Lee, Chi-Kang. “A multiple-path routing strategy for vehicle route guidance systems.” Transportation Research Part C: Emerging Technologies 2.3 (1994): 185-195.
[4] Belwariar, Rachit. “A* Search Algorithm.” GeeksforGeeks, 1 Feb. 2021, www.geeksforgeeks.org/a-search-algorithm/.
[5] Watkins, Christopher JCH, and Peter Dayan. “Q-learning.” Machine learning 8.3-4 (1992): 279-292.
[6] Sutton, Richard S., and Andrew G. Barto. “Reinforcement learning: An introduction.” MIT press, 2018.
[7] Lopez-Guede, Jose Manuel, Borja Fernandez-Gauna, and Manuel Grana. “State-action value function modeled by ELM in reinforcement learning for hose control problems.” International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems 21.supp02 (2013): 99-116.
[8] Shyalika, Chathurangi. “A Beginners Guide to Q-Learning.” Medium, Towards Data Science, 16 Nov. 2019, towardsdatascience.com/a-beginners-guide-to-q-learning-c3e2a30a653c.
[9] L. Littman, Michael, “Markov games as a framework for multi-agent reinforcement learning” Machine Learning Proceedings 1994, Morgan Kaufmann, 1994, ISBN 9781558603356