Lab 4 - A* Algorithm

Songyou Peng
psy920710@gmail.com

I. INTRODUCTION

As a new learning trend, reinforcement learning benefits many research area and specially enables a robot to autonomously discover an optimal behaviour through iterative trial and error [1]. In reinforcement learning, an agent walks in an unknown environment and tries to optimize its long-term return by performing actions and receiving rewards. The main task is to check how the current action can affect the future rewards [2]. Q-learning, as one of the simplest but efficient reinforcement learning algorithm, can learn the optimal policy or the best action in each position to the goal.

Throughout the lab, we will firstly discuss the details of our work and illustrate the results in different scenarios. And then the way that we calculate the evolution of the effectiveness will also be introduced. Problems that we faced and final remarks will be put in the end.

II. IMPLEMENTATION

A. Q-Learning algorithm

The algorithm is split to a large number of episodes and the learning process is repeated within each episode. An episode includes 50 iterations in our cases. In every iteration, the agent starts with a random free cell and moves depending on a ε-greedy strategy. And then, with the chosen action, we can update the action value function Q and current state and loop to the next iteration. If the next state is exactly the goal point, an episode finishes and next one starts.

First of all, The initial position ought to be chosen at the beginning of every episode. One thing needs to be careful is to ensure the place should be neither inside obstacles nor the goal point. Neither cases don't obey the algorithm so it may finally lead to a wrong or weird result. After initializing the starting position, the iterative learning procedure starts.

Up, down, left and right are the four actions that an agent can use. In order to know where the agent should move, a ε-greedy policy is applied. A random float number between 0 and 1 is created, and the neighbour with the largest value in Q function is chosen as the next position if it is larger than the given ε. Otherwise, we randomly choose an action out of four. After moving, we have the next state s', which contains three possibilities:

1) Goal
2) Obstacle
3) Free space

For all the three cases, a learning process or value updating is applied. The formulation is shown as followed:

$$Q(s,a) = Q(s,a) + \alpha[r + \gamma \max_{a'} Q(s',a') - Q(s,a)] \tag{1}$$

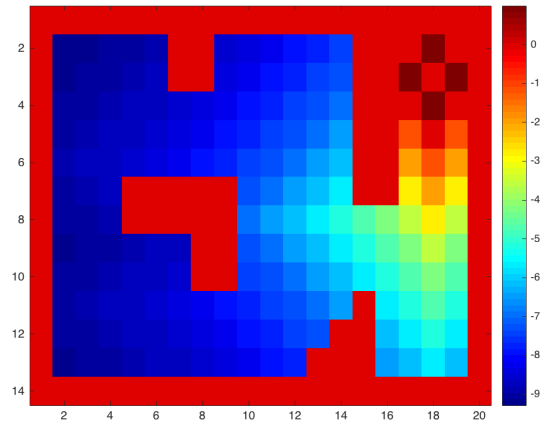
where α is the learning rate, r is the reward in dynamic function, γ is the discount factor, while a' is the actions for the next state. As we can see in this equation, the agent receives the future value of $r + \gamma \max_{a'} Q(s',a')$, which is the current reward plus the discounted future value.

If the next state is the goal point, reward r is given 1, while -1 is assigned to another two cases. Also, when the obstacle is found in the next state, the state will not be updated.

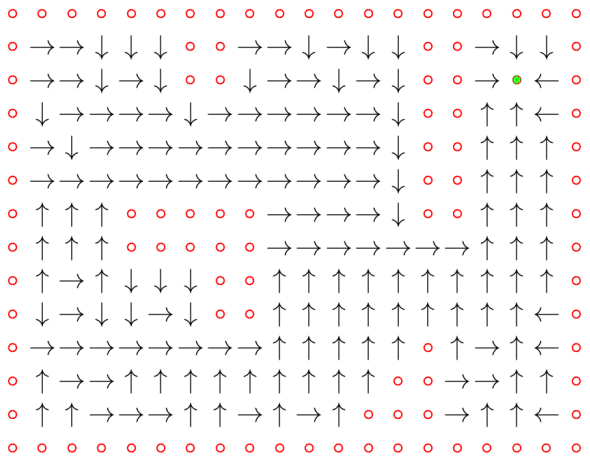
When the learning period finishes, we acquire four Q corresponding to 4 actions. The final state value functions V is the maximum value in each cell, which also represents the best action for that cell. The graphical representation of V and the best policy is shown in Fig. 1.

B. evolution of the effectiveness

For the sake of checking if the algorithm works properly, a test should be introduced. Specifically,



(a) State value function



(b) Optimal policy

Fig. 1. Illustrations for the results acquired from Q-learning.

every 200 episodes, we get the average of the rewards, which equals to the negative value of the average number of iterations. If the average decreases with respect to the increment of episodes, we can say it is converging and the algorithm works correctly. The evolution of the effectiveness is shown in Fig. 2:

We also try two other scenarios, which have been shown in Fig. 3.

C. Faced problem

There was a problem that we spent much time debugging. When the next state is in the obstacle, the s' should be kept the same as s in Eq. (1). However, we didn't notice this part and calculated $\max_{a'} Q(s', a')$ using s'. This actually means that the movement to obstacles is allowed for updating

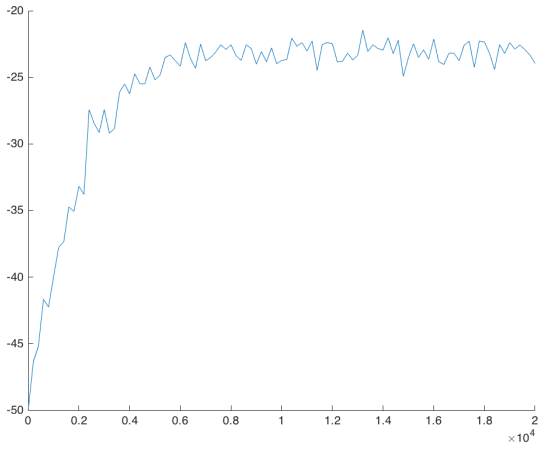


Fig. 2. Evolution of the effectiveness

action function. When we compare the Fig. 4 with Fig. 1 and 2, we notice almost indeed all the optimal actions point to the obstacles and the algorithm does not converge.

III. Final Remarks

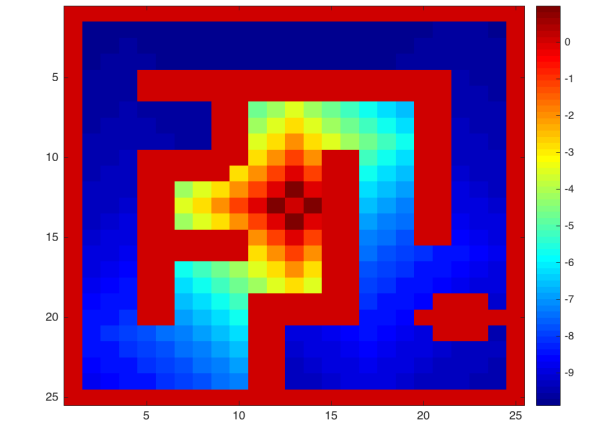
In this lab, the Q-learning algorithm is implemented for robotics path planning and successfully tested in different scenarios. The algorithm promises giving an optimal policy if all the parameters are given properly.

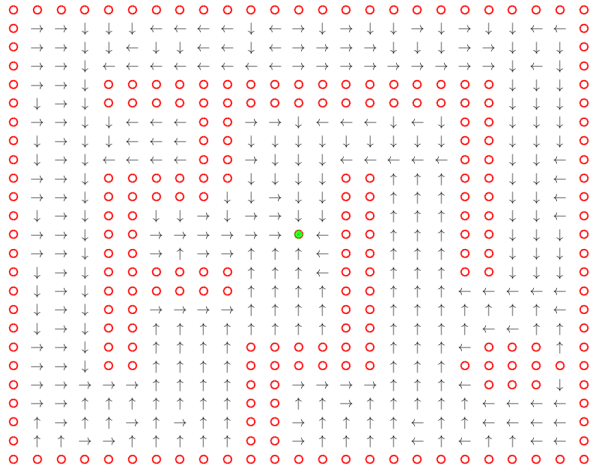
Compared with other path planning algorithm, one main drawback is time-consuming. If the given map is large, it costs a really long time to calculate the optimal policy. Also, when we test larger environment, the algorithm will not give correct policy if the number of the episode is not big enough. Correspondingly, increasing the number of the episode will also increase the computational time.

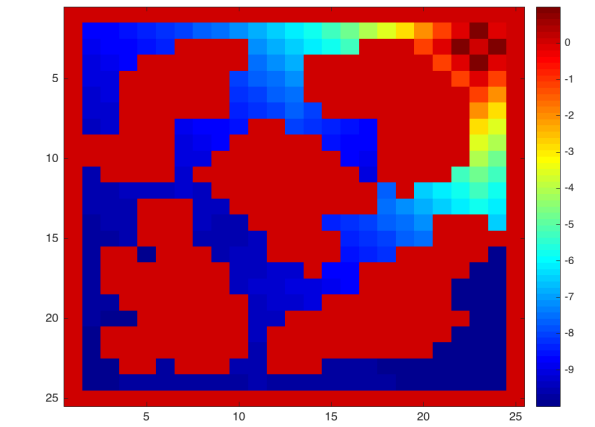
References

[1] Kober, Jens, J. Andrew Bagnell, and Jan Peters. "Reinforcement learning in robotics: A survey." The International Journal of Robotics Research (2013): 0278364913495721.

[2] Even-Dar, Eyal, and Yishay Mansour. "Learning rates for Q-learning." The Journal of Machine Learning Research 5 (2004): 1-25.







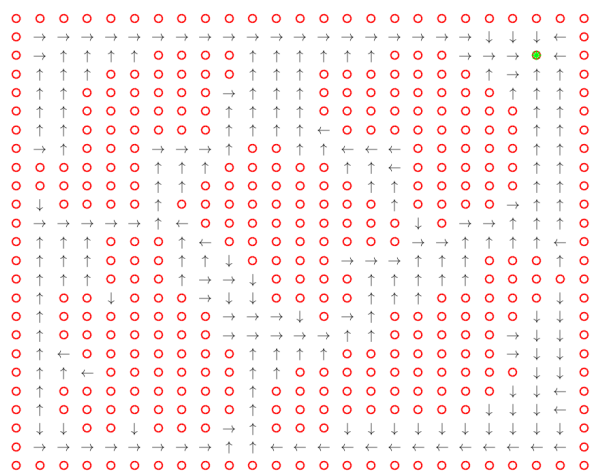
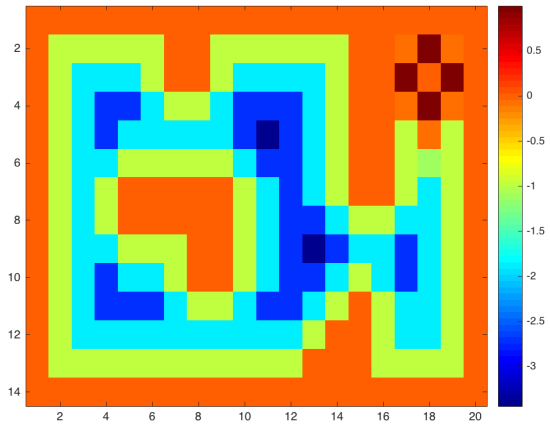
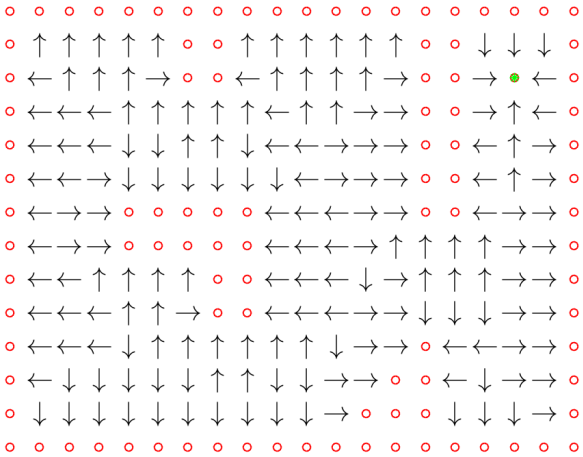


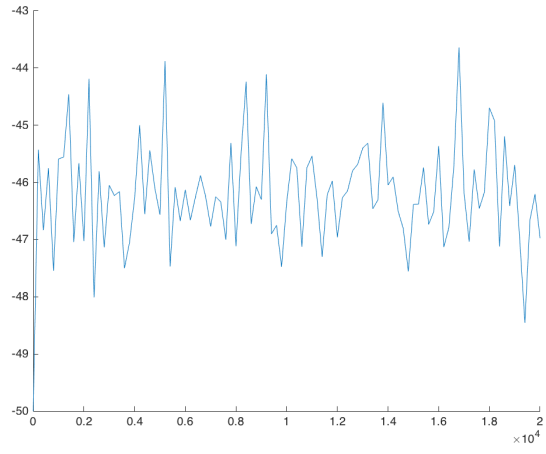
Fig. 3. Illustrations for Q-learning results in other scenarios.



(a) State value function



(b) Optimal policy



(c) Evolution of the effectiveness

Fig. 4. Illustrations for the results acquired from the faced problem