Artificial Intelligence Assessment

Introduction

This report is to solve a "frozen lake" problem, which needs to find a safe and optimal path from start point to goal point on the frozen lake and avoid all holes on its surface. Begin with analyze the PEAS, then explains the method and theory of three kinds of agent (i.e. random agent, simple agent and reinforcement agent), and briefly introduces how they implement. In addition, analysis the output of three agent for a specifically problem then compare their performance.

1. Performance measure

The agent needs to find a path from start point to goal and avoid dropping in holes. The shortest way will be the optimal. In addition, if the agent uses less iteration and less episode, the performance will be better.

2. Environment

A frozen lake with many holes on the surface. An 8*8 grid is used to represent the lake. H means the holes, which is the doom, G is the goal point. They will both be the end of game but with different reward. S is the starting point, F is the frozen surface, both of them are safe.

3. Actuators

The agent can move to four different directions. i.e. up, down, left and right.

4. Sensors

Random agent: None
Simple agent: Oracle
Reinforcement agent: Perfect information about the current state and thus available actions in that state; no prior knowledge about the state-space in general

Method/design

1. Senseless/Random agent

Random agent ignores the percept history and act randomly, therefore, it does not have the ability of learning from the feedback of reward and environment. All it can do is randomly move until get to the goal or drop in a hole.

2. Simple agent

When the state-space is fully observable and discrete, A* could be used to solve the solution. It is one of the best and popular technique used in path-finding and graph traversals. A* picks the node at each step according to a value- ‘**f**’ which is a parameter equal to the sum of two other parameters – ‘**g**’ and ‘**h**’. At each step it picks the node/cell having the lowest ‘**f**’, and process that node/cell.

$$f(n)=g(n)+h(n)$$

g = the movement cost to move from the starting point to a given square on the grid, following the path generated to get there.
h = the estimated movement cost to move from that given square on the grid to the final destination. i.e. heuristic

3. Reinforcement agent

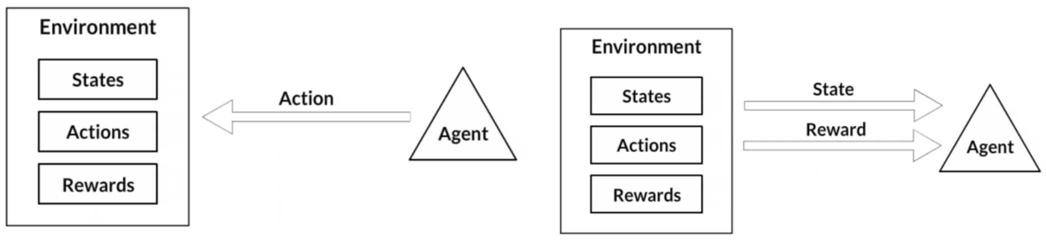


Figure 1. The process of Q-learning

Here, Q-learning is used as a reinforcement agent. Q-learning is the method of finding the optimal solution without prior knowledge only by learning from experience of interaction with environment. When agent take actions on environment, the state and reward will be feedback to the agent. In Q-learning, the agent is so smart that it always intends to take actions which could get higher rewards.

To be specifically, firstly initial the Q table with zero and parameters including gamma and learning rate. Standing on the current state, if it is not the terminal state, randomly choose an action and execute it to obtain next state S’ and reward. Then calculate the Q table with the function:

$$Q(s,a)=Q(s,a)+\alpha(R(s)+\gamma \max a' Q(s',a')-Q(s,a))$$

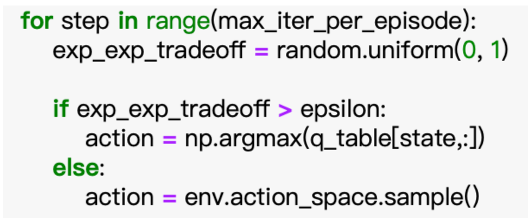
Q table is the core of Q learning, the agent has to continuous update Q table and take actions depends on the value in it. To find the best path, max value will be selected since it gives the higher reward.

Implementation

1. Random agent： use "action = env.action_space.sample()" to randomly move.

2. Simple agent: create a known graph object firstly, then search with the lowest f scores using "my_best_first_graph_searh (problem, f)", and find the best solution by function "my_astar_search(problem, h=None)"

3. Reinforcement agent: Firstly initializing the q_table and parameters, then choose action for next step using：



```
for step in range(max_iter_per_episode):
    exp_exp_tradeoff = random.uniform(0, 1)

    if exp_exp_tradeoff > epsilon:
        action = np.argmax(q_table[state,:])
    else:
        action = env.action_space.sample()
```

Figure 2. Code of choosing action

After getting new information from new state, updating the Q table with bellman equation:

```
q_table[state, action] = q_table[state, action] * (1 - learning_rate) + \
learning_rate * (reward + discount_rate*np.max(q_table[new_state, :]))
```

Evaluation

To evaluate the performance of three agent, compare their results of a same problem. i.e. problem 7

1. Random agent

In this experiment, initialize the episode of 2000 and iteration with 99, the result shows that the success rate was quite low. To calculate the score, sum the rewards in every episode and divide it by number of episodes.

From the results below, we can see that only 22 success in 2000 episodes, the performance is not good.

```
Problem_7      Number of Success: 22      Score: 0.011
```

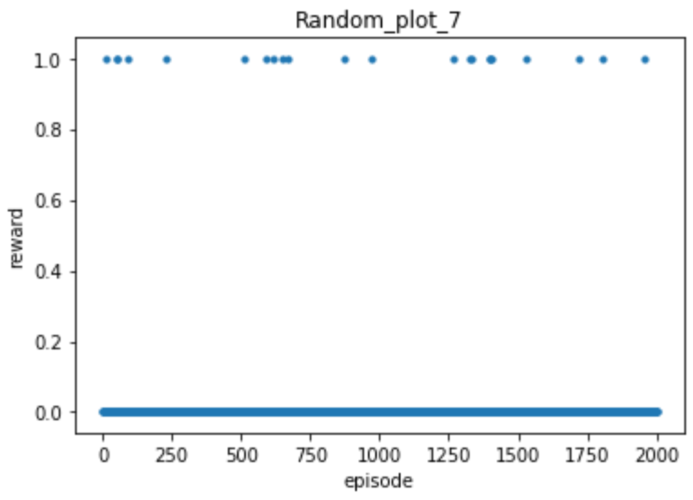
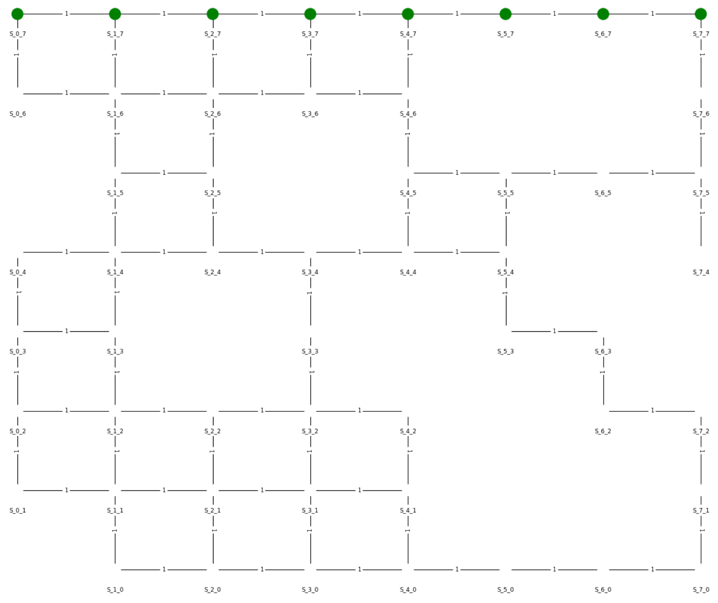


Figure 3. Scatter plot of problem 7 for random agent

2. Simple agent

A* works in a known environment, so it is meaningless to collect the rewards since it always finds the optimal solution. However, the iteration A* using to get to the goal could be useful to evaluate its performance. In problem 7, A* takes 157 iterations to find the final optimal path.



```
 problem_id :   7
 Identified goal state:   <Node S_7_7>
 Number of Steps:   7
 Solution trace:   [<Node S_7_7>, <Node S_6_7>, <Node S_5_7>, <Node S_4_7>, <Node S_3_7>,
<Node S_2_7>, <Node S_1_7>, <Node S_0_7>]
 iteration used:   157
```

Figure 4. Result of simple agent

3. Reinforcement agent

Q learning is a model-free method, which mainly depend on Q table to make decision. The results change as the parameters (i.e. learning rate, gamma, epsilon) and initialization change. The picture below is the average rewards over 100 episodes, it is clear that after increasing quickly, the rewards fluctuate in a range and converge to a certain reward value, which means that after many times of episodes, it learned from it and select the better way later. In addition, the average steps over 100 episodes ranges between 60 to 80 steps after the sharply fluctuate at the beginning.

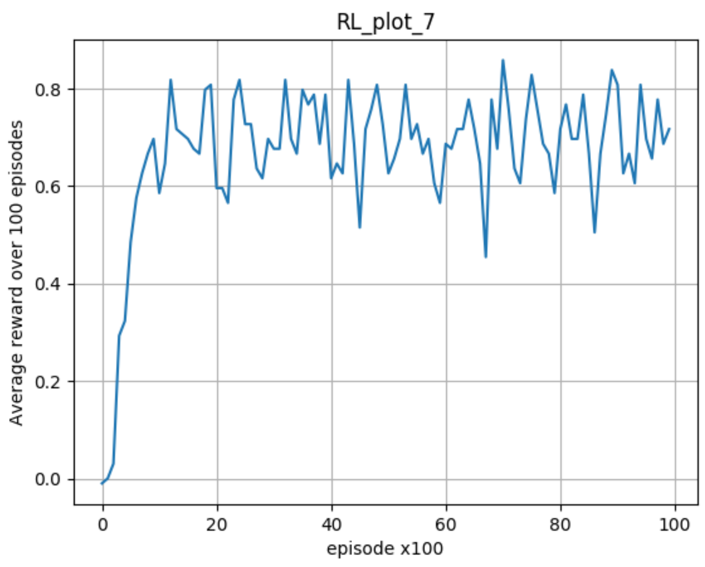


Figure 5. Average reward vs episode plot for problem 7

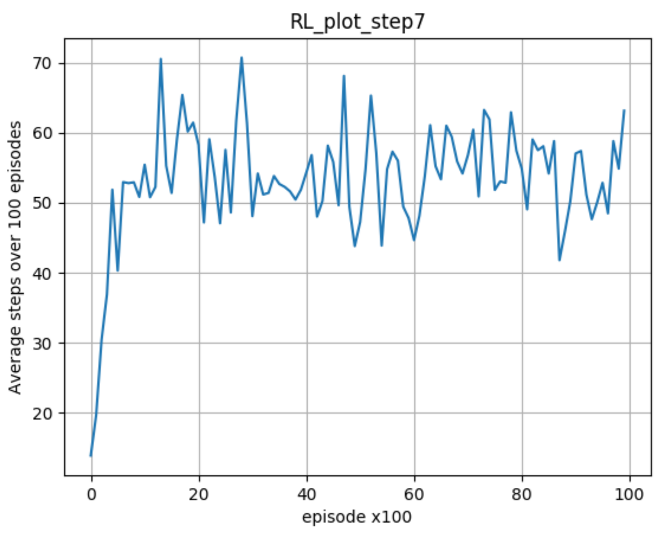


Figure 6. Average steps vs episode plot for problem 7

Comparison:

From the results above, it is clear that the performance of random agent is the worst one, only around 1% even lower proportion of episodes get to the goal points, and it is not sure that the solution is the optimal path. A star could always find the best way, but it needs to know the full environment and details. Reinforcement agent finds the best path through thousands of episodes' experience, it takes less steps to the goal than A* and get obviously higher performance measure than random agent.

Discussion

Both A* and Q learning can solve the "frozen lake" problem, but A* required prior knowledge of the environment, which is not always satisfied in real world, A* star might be more suitable only in some specific problems. However, Q learning could find the solution without prior knowledge, and learning from its own history smartly.