Markov Decision Process

Xujian Liang, GTID: 903421366

1. Introduction

In this assignment, I will implement several reinforcement learning algorithms to two Markov Decision Processes (MDPs). The algorithms are Value Iteration (VI), Policy Iteration (PI), as well as Q-Learning (QL). Python and Java are used for the plotting in this assignment.

2. MDPs Description

The two MDPs are modified from the Grid World of BURLAP (http://burlap.cs.brown.edu/ and https://github.com/XuetingChen/cs7641-assignment-4). The original Grid World (Figure 1) is a 2-D field with the agent (brown ball) on the lower left grid, and the target (blue square) on the upper right grid. The black grids represent obstacles which the agent cannot pass through. The agent has to travel to the target grid bypassing the obstacles with four possible movements: up, down, left and right. A cost function is reinforced for each movement of the agent, such that -1 point, for instance, is obtained for each step. However, if the agent successfully reaches the target, it will receive a large and positive reward of, for example, 100 points. To simulate real-world situation, the agent has a probability of P for each movement. The user can specify iteration numbers for each experiment. Here I use 1000 iterations for each experiment.

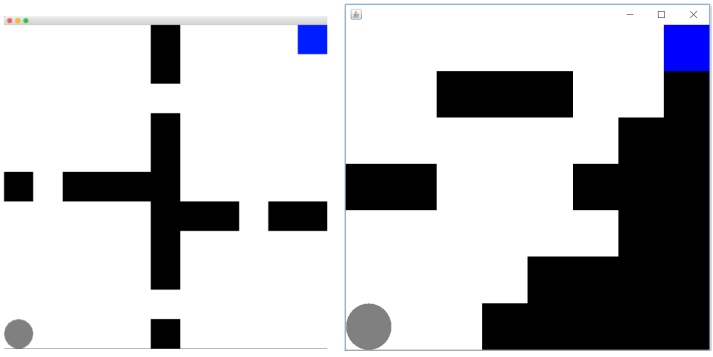


Figure 1: original Grid World set up. Figure 2: Small state grid set up simulating oceanic glider retrieve process.

2.1. Oceanic Glider Retrieve Process

A glider is an underwater torpedo-like instrument that can “swim” in the ocean and measure various variables with different sensors. Example of underwater gliders can be found at

https://en.wikipedia.org/wiki/Underwater_glider. Gliders are controlled to travel in a vertically zig-zag path to measure water properties from surface to bottom of the ocean (or vice-versa). The propel power of the gliders is very limited, so one has to very careful when piloting the gliders to avoid being trapped by obstacles and strong currents in the ocean, especially during glider retrieve process. Near the shore, the topography usually has an angle, so the subsurface glider has to avoid plunging into the bottom near the shore. Meanwhile, there could be strong near-shore currents above the glider, which can push away the glider and causing losing of the glider, so the strong currents above the glider should be avoided.

The MDP simulates the glider retrieve process near the shore of the ocean (Figure 2), with slope of shore on the right, and some obstacles above the glider (simulating strong currents). The glider on the lower left grid has to travel to the upper right grid, without plunging into the slope or hitting the obstacles above. As a small number of states MDP, the grid has a size of 7 by 8 (56 grid). In this experiment, the cost function is -1 for each step. The target reward is 100.

2.2. Maze

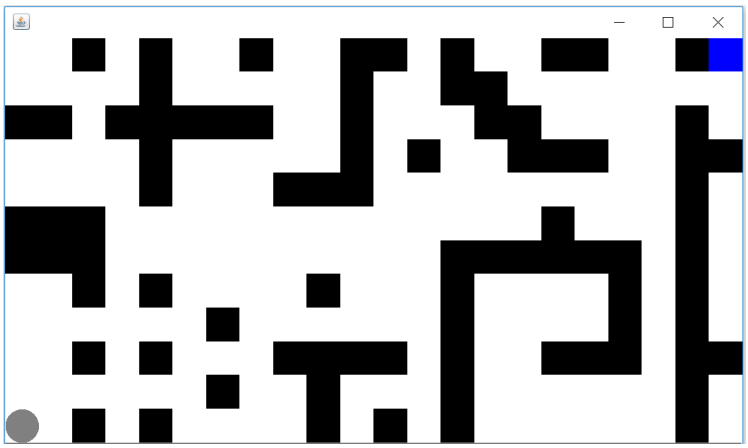


Figure 3: Large state grid set up simulating the maze.

The second MDP simulates a large maze. An agent has to travel from the lower left grid to the upper right grid (the target) without hitting the black walls (Figure 3). This problem is interesting because it could be applied to many real-world situations such as search and rescue missions in the woods or mountains, or finding enemies in a war zone without hitting traps, or finding missing pets in the cities. As a large number of states MDP, the grid has a size of 12 by 22 (264 grid in total). In this experiment, the cost function is also -1 for each step. The target reward is 100.

3. Methodology overview

3.1. MDP

A Markov decision process(MDP) is a stochastic process in a discrete time scale, which helps the agent to make decisions in situations with partially random and partially controlled results. A MDP is completely defined with the flowing terms: States, Action, Transition, Reward and Discount Factor. With MDP, policies can be generated through iteration process to help the agent to make important decisions according to the rewards gained.

3.2. Value Iteration

VI iterates by applying the Bellman equation to evaluate the state at each iteration. It starts with initial value V_0, and then calculate the next value V for the state in the next level by applying the Bellman equation until convergence.

3.3. Policy Iteration

While value-iteration algorithm keeps improving the value function at each iteration until the value-function converges. Since the agent only cares about the finding the optimal policy, sometimes the optimal policy will converge before the value function. Therefore, another algorithm called policy-iteration instead of repeated improving the value-function estimate, it will re-define the policy at each step and compute the value according to this new policy until the policy converges. Policy iteration is also guaranteed to converge to the optimal policy and it often takes less iterations to converge than the value-iteration algorithms.

Neither policy or value iteration methods are not reinforcement learning algorithms. As long as we know the transition matrix and reward matrix of the environment, both policy and value iteration methods can find an optimal methods for MDP.

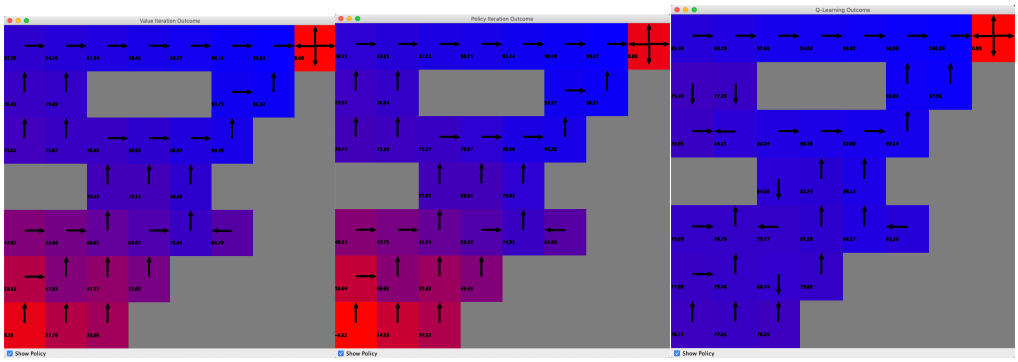
3.4. Q-Learning

QL first assigns a Q-value to every state. It then updates the Q-value using the immediate and delayed rewards at the next iteration. This process iterates until the Q-values converge.

4. Methodology overview

4.1. Glider retrieve process.

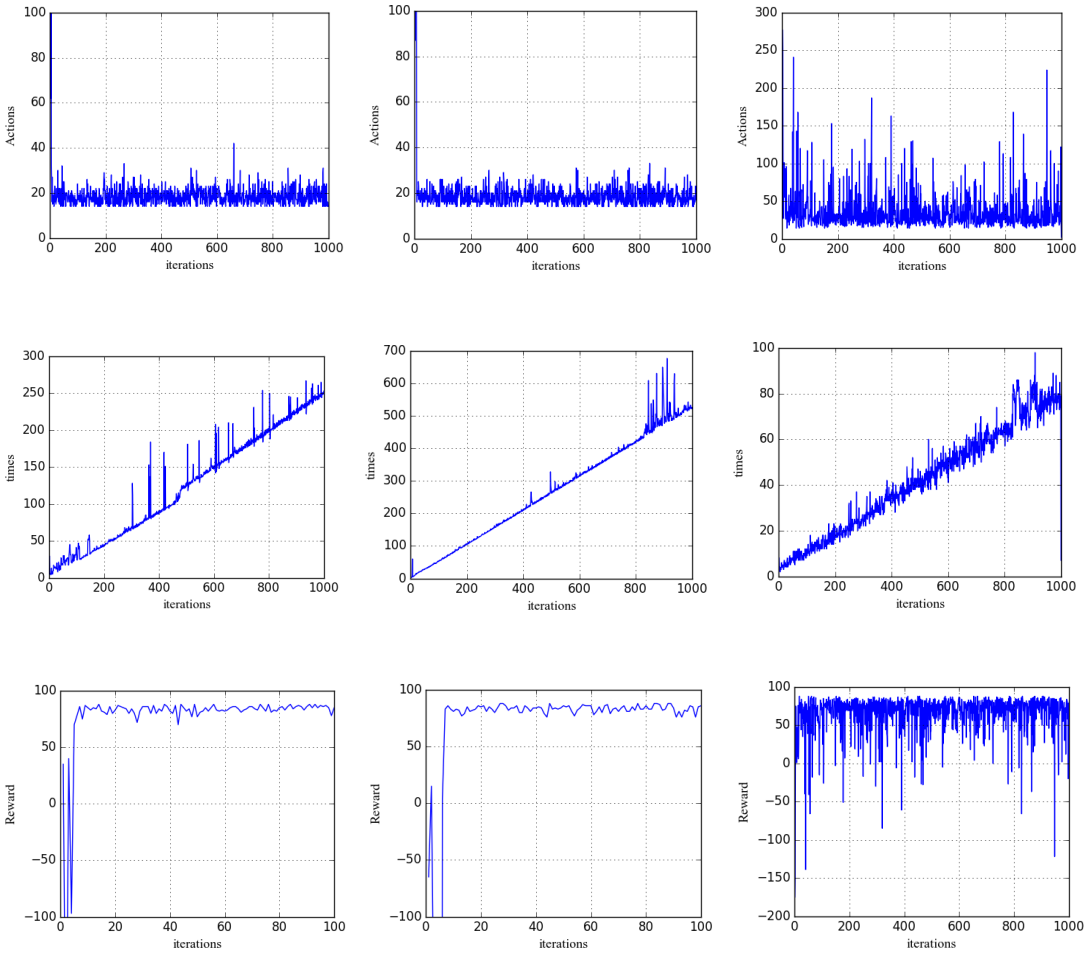
The VI, PI and QL results are presented in below.



VI result(left), PI results(mid), QL result(right)

It can be seen that VI and PI results are similar to each other, probably because of the simplicity of the problem set up. QL is different compared with VI and PI. The reason behind it is that QL try to maximize the total rewards from all iteration steps instead of evaluation at each step.

Below is the number of action vs iteration number, iteration time vs iteration, rewards vs iteration of VI, PI and QL.



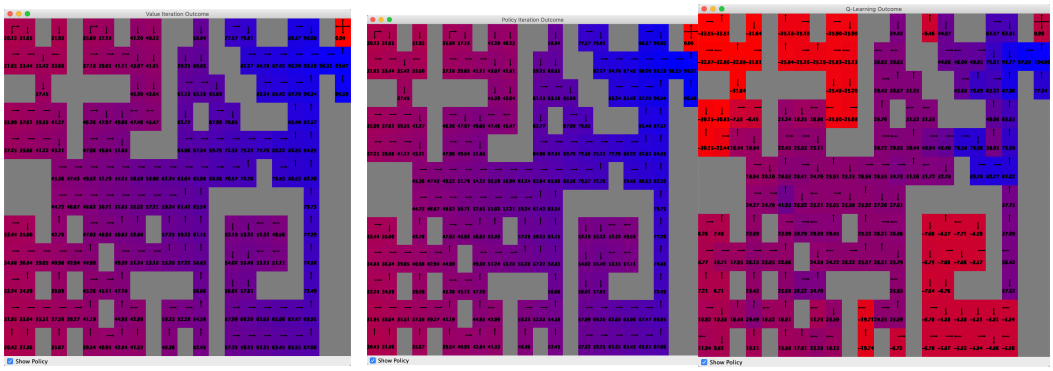
VI result(left), PI results(mid), QL result(right)

It seems the first few iterations require large amount of actions (steps) for both VI and PI, and the

actions decline fast to the level of 20, indicating the model is stabilizing. For QL, there seems to be lots of spikes, indicating QL is not as stable as VI and PI. This instability also shows in iteration time, where the values also have many spikes. From the reward results, one can see VI and PI quickly converge in the first 10 iteration, while QL seems to be difficult to converge, consistent with the instability in actions and time.

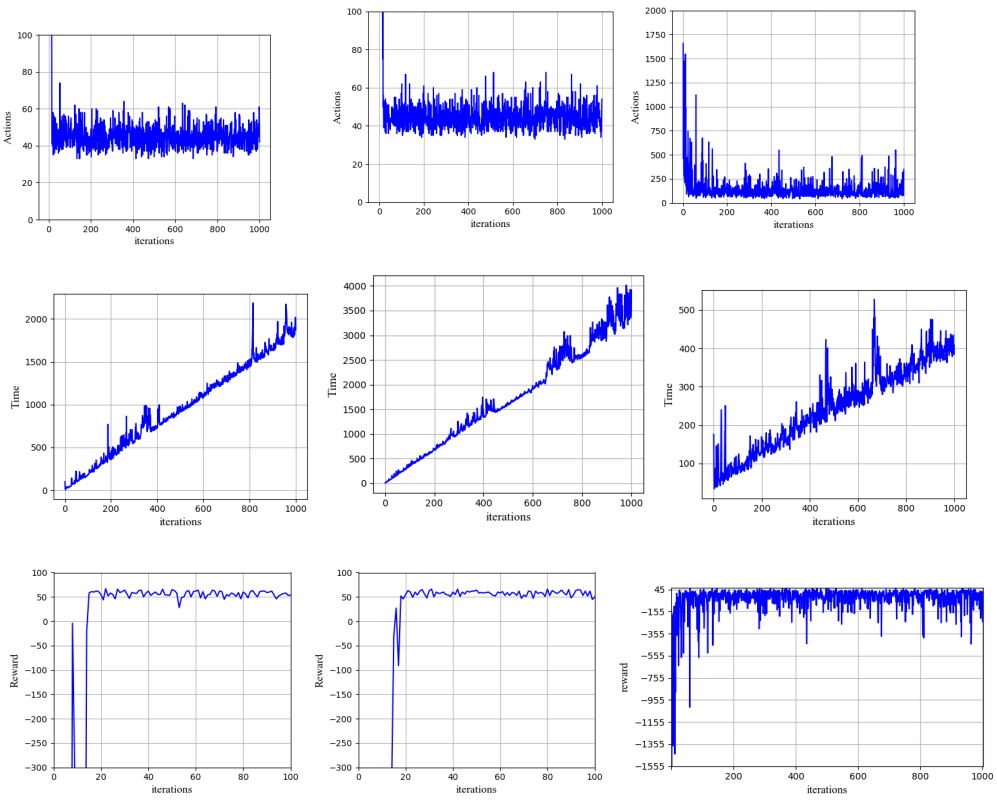
4.2.Maze

The VI, PI and QL results of the Maze case are presented below.



VI result(left), PI results(mid), QL result(right)

Again, VI and PI show similar results, while QL has a different map. The reason behind it is that QL try to maximize the total rewards from all iteration steps instead of evaluation at each step.



Number of action vs iteration number for VI (top left), PI (top middle), QL (top right). Iteration time vs iteration number for VI (middle left), PI (middle middle), QL (middle right). Reward vs iteration number for VI (bottom left), PI (bottom middle), QL (bottom right) for maze.

The number of action vs iteration number, iteration time vs iteration, rewards vs iteration of VI, PI and QL for the Maze case are shown in above. Similar to the glider case, VI and PI need lots of actions in the first few iterations, and the actions decline fast to the level of 50, indicating the model is stabilizing. The action numbers for QL also decrease after few iterations, although the instability still presents with spikes. All three algorithms show a linear increase of time increased iteration. For VI and PI, the algorithms converge fast in the first 20 iteration, which is longer than the glider case. This is because the Maze case has a much large state number (12 by 22) and much more complicated (with many obstacles) than the glider case. The VI and PI converge when the reward stabilizing near 60, while the QL seems to be difficult to converge with reward value less than 0 and with many negative spikes, indicating instability of QL again.

5. Strategies

In the previous section, the exploration strategy employed for the QL algorithm is the EpsilonGreedy strategy, which is a relatively simple strategy. To use this strategy, the user picks a small epsilon number (default 0.1 in previous cases), and then "explores" the rewards in the first few tries. The algorithm then allocates higher probability (a function of epsilon) to the higher reward policies in the following iterations (exploit). This process repeats until there is no further improvement.

The problem with this approach is that if the algorithm explores the first policy with higher rewards compared to the second policy, then it will choose the first policy immediately, regardless if the second policy might give a higher reward in a later time.

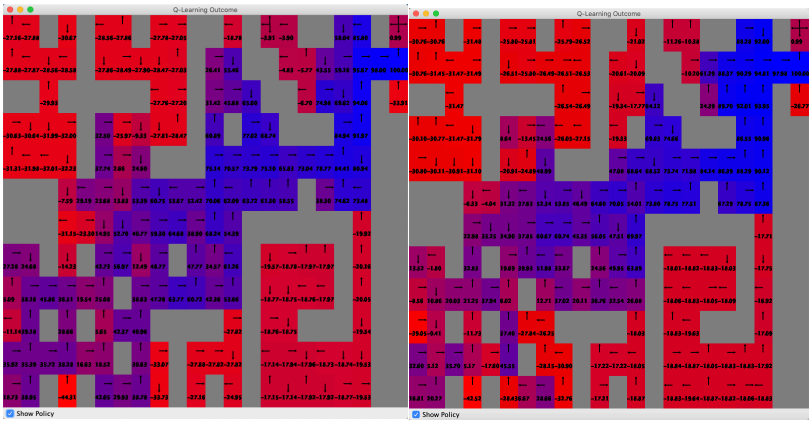
The Boltzmann strategy weights the Q-value for each action and chooses the action with this weighted probability in the following moves. In this way, the action that might have a higher reward in a later time will be taken into consideration. However, no method is perfect. The disadvantage of the Boltzmann strategy is that it might overlook how certain an agent believes the action is option in the real world.

Here we implement both strategies with different parameters.

5.1. Boltzmann Strategy

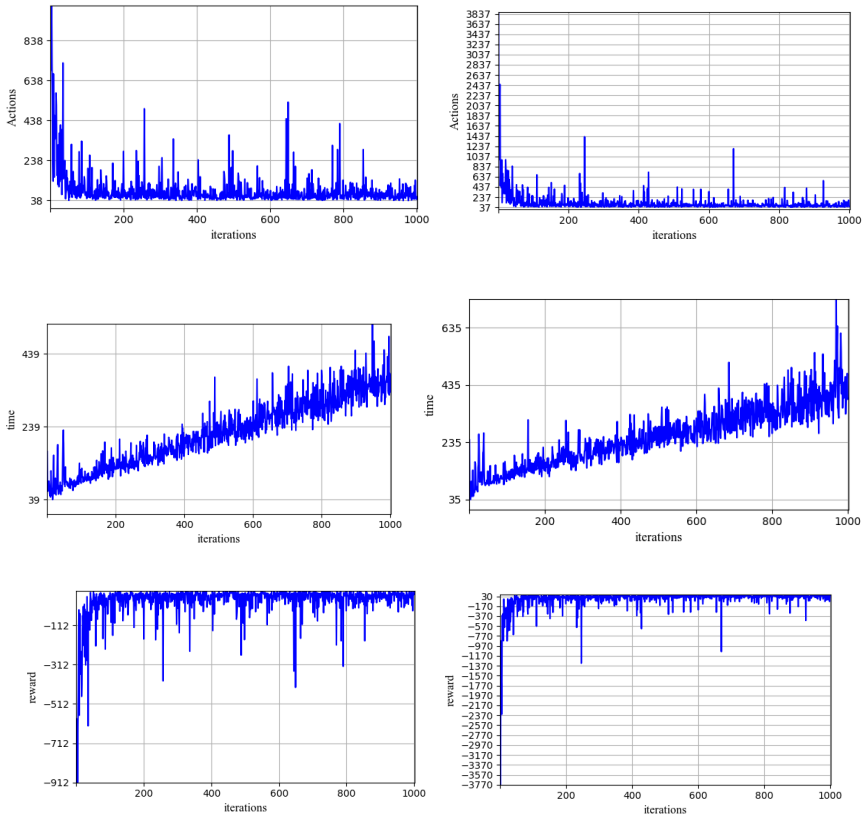
Here I assign the "temperature" (similar to annealing method) value of 0.1 and 0.9 to the Boltzmann

strategy. The results are presented below.



QL Boltzmann strategy results with temperature = 0.1 (left) and temperature = 0.9 (right).

The map distributions show that there is no significant changes in the results with both temperature = 0.1 and 0.9. Both the upper left corner and lower right corner still have low Q-values, indicating the algorithm doesn't learn much from this semi-enclosed areas. The temperature values of 0.1 and 0.9 can both recognize these low activity areas and find the best path to the target.



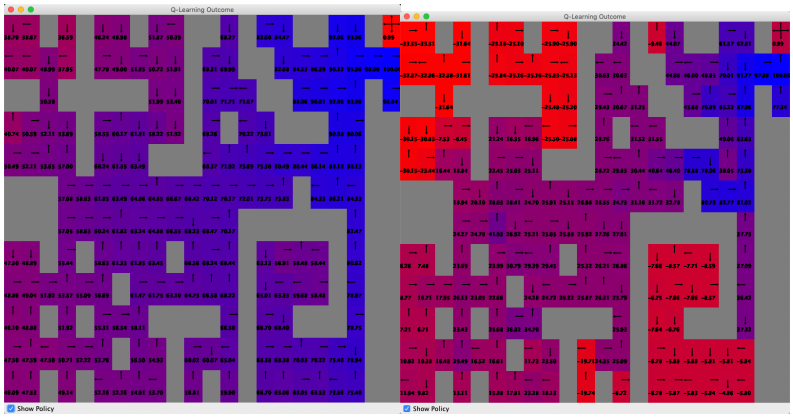
Number of action vs iteration number for QL Boltzmann strategy with temeprature = 0.1 (top left) and temperature = 0.9 (top right); Iteration time vs iteration number for QL Boltzmann with temperature

= 0.1 (middle left) and temperature = 0.9 (middle right); Reward vs iteration number for QL Boltzmann with temperature = 0.1 (bottom left) and temperature = 0.9 (bottom right).

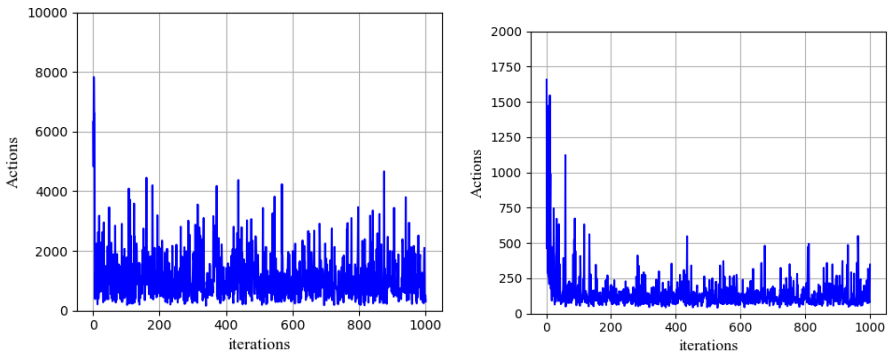
The action, time and reward vs iterations (Figure 9) also show similar results for both temperature = 0.1 and 0.9. Although with temperature = 0.9, the algorithm needs more action and time with increasing iteration numbers. This might be because the temperature value are decreasing in the annealing process. With a high initial temperature, the algorithm will need more time for the temperature to decrease. The rewards also decrease with high temperature value. Therefore, a low temperature value is more suitable here.

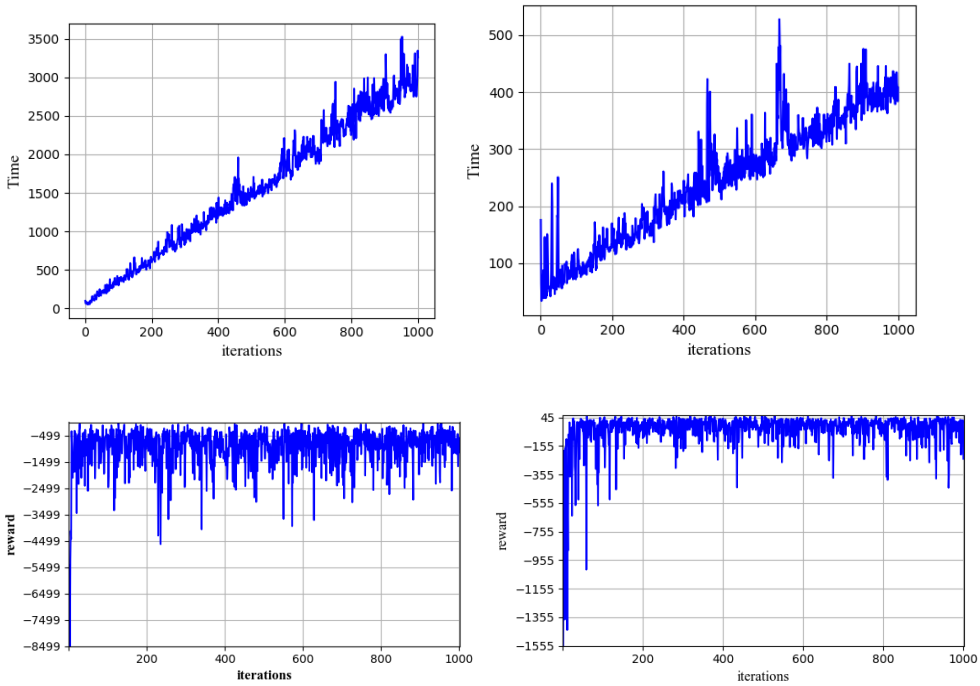
5.2. EpsilonGreedy Strategy

Here I assign the epsilon value of 0.9 and 0.1 to the EpsilonGreedy strategy. The results are presented below:



QL EpsilonGreedy strategy results with epsilon = 0.9 (left) and epsilon = 0.1 (right).





Number of action vs iteration number for QL EpsilonGreedy strategy with epsilon = 0.9 (top left) and epsilon = 0.1 (top right); Iteration time vs iteration number for QL Boltzmann with epsilon = 0.9(middle left) and epsilon = 0.1(middle right); Reward vs iteration number for QL Boltzmann with epsilon = 0.9 (bottom left) and epsilon = 0.1 (bottom right).

With two very different epsilon values, the map results are quite different, especially in the Q-value distribution. The one with epsilon = 0.1 shows low Q-value in the two semi-enclosed areas, while the one with epsilon = 0.9 shows higher Q-values in these two areas, indicating the algorithm does learn a lot when exploring these areas. This is because a high epsilon value forces the algorithm to explore the areas that the agent is not familiar with. As a result of the additional exploration, the algorithm need more action, time during the iteration. The resulting rewards are also low due to the cumulation of additional explore actions. Therefore, in this case, it seems like the most appropriate value for epsilon is a lower value of 0.1.

5.3. Comparison of Boltzmann and EpsilonGreedy Strategies

Though Boltzmann and EpsilonGreedy are two different strategies for the QL algorithm, they show some interesting qualities in the results. The lower temperature for Boltzmann and low eplision for EpsilonGreedy both need lower action and lower times. The higher value of epsilon for Epsilon Greedy, however, takes much more actions and time for each iteration due to the additional exploration in the semi-enclosed areas than the high value of the temperature for the Boltzmann algorithm.

No matter which strategy, we can still see instability of the QL in action, time and rewards, which means the stability may be not related to the strategy. From my results, I think it's part of because of the combination of the complexity of the maze and QL algorithm itself. Because QL has to judge the overall reward rather than doing the evaluation at every iteration, it has increased its ability to get stable results.

Though the EpsilonGreedy strategy with higher value of epsilon require much more time for each iteration, the linearity of the time increase doesn't change, which is similar to the Boltzmann strategy. This means both algorithms explore the space in a linear space.

6. Conclusions

In this project, I implemented the VI, PI and QL reinforcement learning algorithms to two MDPs. Comparisons are made between these algorithms. It looks like VI and PI produce similar results. They also consume similar actions, time and obtain similar rewards in both MDPs. This could be because they are solving the exact same problem in the same space. Both VI and PI converge faster than QL.

QL produces much more different results compared with VI and PI, because QL consider the overall reward paid out, rather than evaluating the values at each iteration. QL seems to have instability in both MDPs.

QL can employ many strategies. Here I used Boltzmann and EpsilonGreedy strategies. I also adjusted the temperature value for Boltzmann strategyand the epsilon value for the EpsilonGreedy strategy. It turned out that a higher value of epsilon for the EpsilonGreedy strategy can force the QL to explore the areas that it is not familiar with, causing additional exploration time, action and reduced rewards. A higher value of temperature also increases the time and action for the Boltzmann strategy, but the time and action didn't increase as much as the EpsilonGreedy with high epsilon value, probably because QL with Boltzmann strategy doesn't need to explore those semi-enclosed areas as much as the QL with EpsilonGreedy strategy.

Although I don't think the instability of the QL is related to the strategies I used, I think it will be interesting to observe the results of QL with other strategies like the Random Policy or Greedy Deterministic strategies.