Assignment 4: Markov Decision Processes

1. Markov Decision Processes

In this report, I explored the Markov Decision Processing, which is different from the function approximation in the previous assignments. In mathematics, a Markov decision process (MDP) is a discrete-time stochastic control process. It provides a mathematical framework for modeling decision-making in situations where outcomes are partly random and partly under the control of a decision-maker. MDP is useful for studying optimization problems solved via dynamic programming.

A Markov decision process is a 4-tuple (S,A,P_a,R_a)}, where:

- S is a set of states called the state space;
- A is a set of actions called the action space (alternatively, A_s is the set of actions available from state s);
- P_a is the probability that action a in state s at time t will lead to state s' at time t+1;
- R_a is the immediate reward (or expected immediate reward) received after transitioning from state s to state s', due to action a.

The state and action spaces may be finite or infinite, for example, the set of real numbers. Some processes with countably infinite state and action spaces can be reduced to ones with finite state and action spaces. A policy function π is a (potentially probabilistic) mapping from state space (S) to action space (A).

In this report, I will use the grid world to represent the MDP problems. It's a two-dimension plan, where the agent can decide which direction to move and can face obstacles. The agent starts at the bottom left square and it can choose four actions (up, left, right, down). The determination state is the blue block on the up-right corner, which also has the biggest reward of 100. During the search for the big reward, each step will cost -1 reward, and sometimes there can be a trap (-100 reward). The agent should explore the grid world and maximize the total reward. While each action is probabilistic, when the agent chooses to move in one direction, it has 80% chance to go in that direction, and 10% chance to go in each perpendicular direction.

2. Two interesting MDP

I created two MDP, the first one (figure on the left) is the easy one, which is an MDP of small number of states. It's a simple maze that has two paths, basically, one path leads to big reward and the other lead to a painful trap. This requires the agent to make the right choice at the start state.

The second MDP (figure on the right) is the hard one, which has a large state. I want to use this MDP to simulate the process that a robot is searching for the charger in the house. the agent can explore the house and choose different paths along with the search, sometimes it will go into the wrong room and fall into traps, or choose the right way and find the charger. This will test if the agent can find the shortest path to maximize the reward.

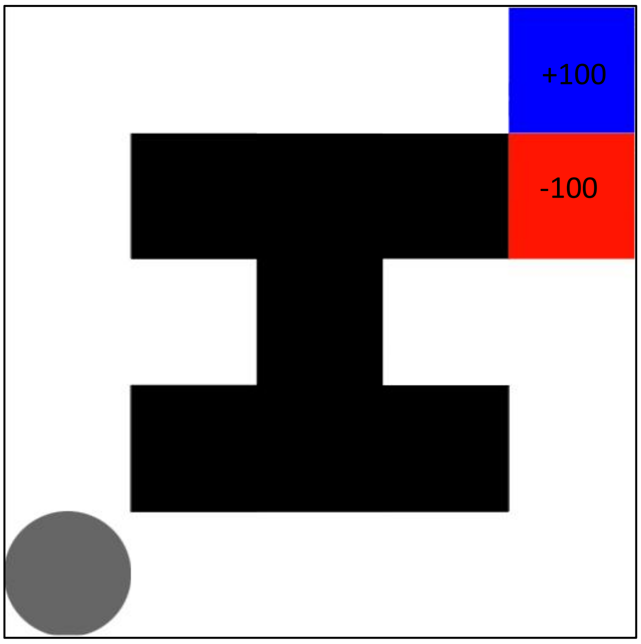

Figure 1. Small MDP(5X5)

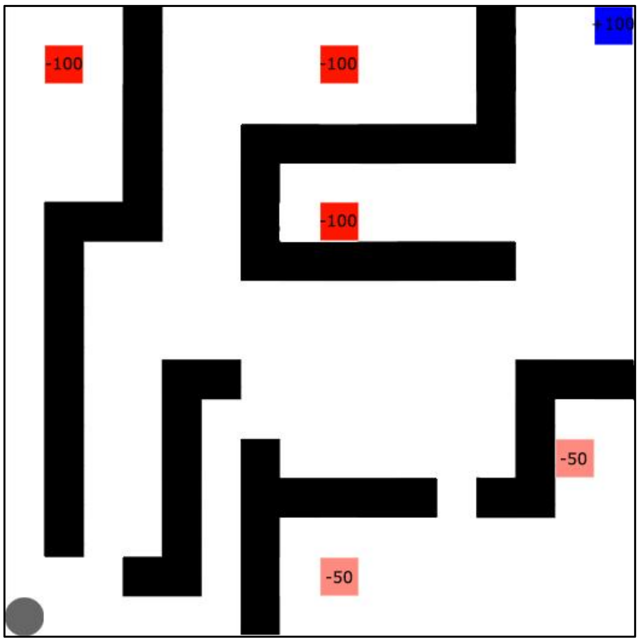

Figure 2. Large MDP(16X16)

3. Solve The MDP

To solve these two MDP, I researched three algorithms: Value Iteration, Policy Iteration, and Q-Learning. Let me start with the introduction of the three algorithms:

Value Iteration

Value iteration is a method of computing an optimal MDP policy and its value. Value iteration starts at the "end" and then works backward, refining an estimate of either Q* or V*. There is really no end, so it uses an arbitrary end point. Let Vk be the value function assuming there are k stages to go and let Qk be the Q-function assuming there are k stages to go. These can be defined recursively. Value iteration starts with an arbitrary function V0 and uses the following equations to get the functions for k+1 stages to go from the functions for k stages to go:

$Q_{k+1}(s,a) = \sum_{s'} P(s'|s,a) \, (R(s,a,s') + \gamma V_{k(s')})$ *for* $k \geq 0$

$V_{k(s)} = max_a \, Q_{k(s,a)}$ *for* $k > 0$

Policy Iteration

In policy iteration, we start by choosing an arbitrary policy π. Then, we iteratively evaluate and improve the policy until convergence:

We evaluate a policy π(s) by calculating the state value function V(s):

$$V(s) = \sum_{s',r'} p(s',r|s,\pi(s))[r+\gamma V(s')]$$

Then, we calculate the improved policy by using one-step look-ahead to replace the initial policy π (s):

$$\pi(s) = arg\max_a \sum_{s',r'} p(s',r|s,a)[r+\gamma V(s')]$$

In the beginning, we don’t care about the initial policy π0 being optimal or not. During the execution, we concentrate on improving it on every iteration by repeating policy evaluation and policy improvement steps. Using this algorithm we produce a chain of policies, where each policy is an improvement over the previous one:

$$\pi_0 \xrightarrow{\mathrm{E}} v_{\pi_0} \xrightarrow{\mathrm{I}} \pi_1 \xrightarrow{\mathrm{E}} v_{\pi_1} \xrightarrow{\mathrm{I}} \pi_2 \xrightarrow{\mathrm{E}} \cdots \xrightarrow{\mathrm{I}} \pi_* \xrightarrow{\mathrm{E}} v_*$$

Q-Learning

I choose to explore Q-learning more, Q-learning is a model-free reinforcement learning algorithm to learn the value of an action in a particular state. It does not require a model of the environment (hence "model-free"), and it can handle problems with stochastic transitions and rewards without requiring adaptations.

Before learning begins, Q is initialized to a possibly arbitrary fixed value. Then, at each time t the agent selects an action at, observes a reward rt, enters a new state st+1 (that may depend on both the previous state st and the selected action), and Q is updated. The core of the algorithm is a Bellman equation as a simple value iteration update, using the weighted average of the old value and the new information:

$$Q^{new}(s_t,a_t) \leftarrow \underbrace{Q(s_t,a_t)}_{\text{old value}} + \underbrace{\alpha}_{\text{learning rate}} \cdot \overbrace{\Bigg(\underbrace{\underbrace{r_t}_{\text{reward}} + \underbrace{\gamma}_{\text{discount factor}} \cdot \underbrace{\max_a Q(s_{t+1},a)}_{\text{estimate of optimal future value}}}_{\text{new value (temporal difference target)}} - \underbrace{Q(s_t,a_t)}_{\text{old value}} \Bigg)}^{\text{temporal difference}}$$

There are some important variables in Q-learning:

Learning rate:

The learning rate α or step size determines to what extent newly acquired information overrides old information. A factor of 0 makes the agent learn nothing (exclusively exploiting prior knowledge), while a factor of 1 makes the agent consider only the most recent information (ignoring prior knowledge to explore possibilities).

Discount factor :

The discount factor γ determines the importance of future rewards. A factor of 0 will make the agent short-sighted by only considering current rewards, while a factor approaching 1 will make it strive for a long-term high reward.

Initial conditions:

Since Q-learning is an iterative algorithm, it implicitly assumes an initial condition before the first update occurs. High initial values can encourage exploration: no matter what action is selected, the update rule will cause it to have lower values than the other alternative, thus increasing their choice probability. The first reward r can be used to reset the initial conditions.

Epsilon:

Epsilon is related to the epsilon-greedy action selection procedure in the Q-learning algorithm. In the action selection step, we select the specific action based on the Q-values we already have. The epsilon parameter introduces randomness into the algorithm, forcing us to try different actions.

Convergence Criteria:

For Q-learning, I set the convergence criteria to be 0.25 to exam the result better.

4. Small MDP

Value Iteration & Policy Iteration (Convergence criteria 10^-6)

First I tried value iteration and policy iteration on the simple grid world. Figure 3 below are the convergence of the two algorithms. I ran 150 times for value iteration and policy iteration, each algorithm will stop iterating once reached the convergence criteria. I set the convergence criteria as convergence delta < 10^-6. We can see from the chart that policy iteration converges faster than value iteration, which can be explained as that the value iteration combines two phases of the policy iteration (one phase evaluates the policy, and the other one improves it) into a single update operation, the function runs through all possible actions at once to find the maximum action value.

Figure 4 shows how much time each iteration takes. As the iteration number goes up, more time is needed to compute. For each iteration, the policy iteration took slightly more time than the value iteration. Policy iteration needs to run two phases, evaluates the policy and the improves it, so it will take more time.

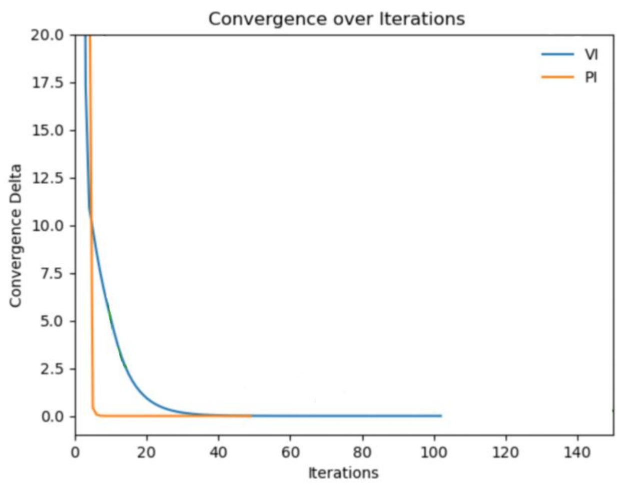

Figure 3

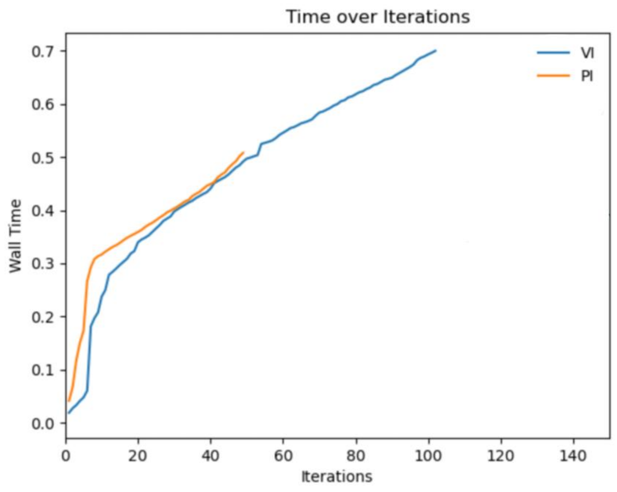

Figure 4

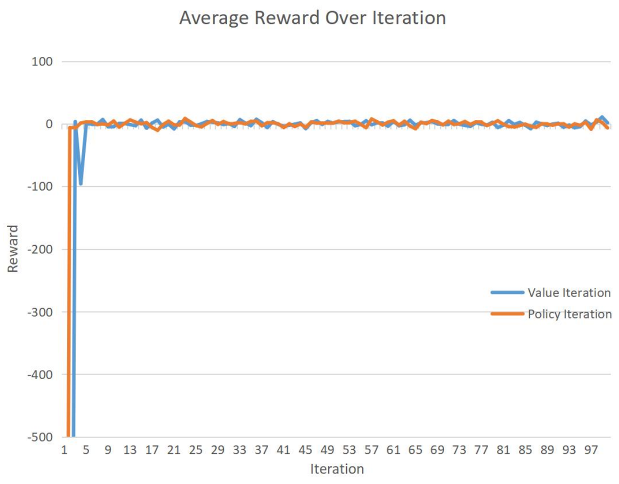

Figure 5

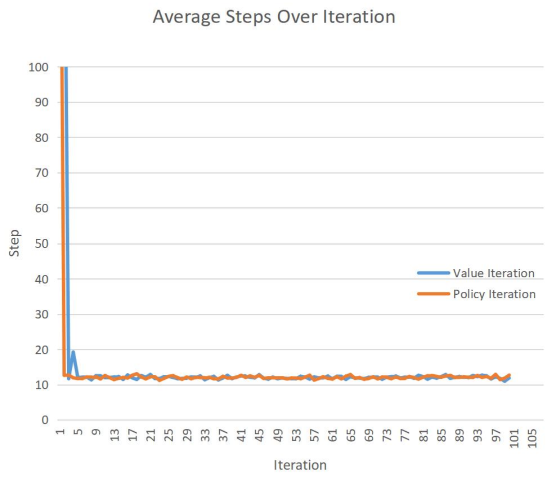

Figure 6

Figure 5 shows the average reward of value iteration and policy iteration. We can see both algorithms didn’t figure out the right policy and wondered in the grid world which caused big negative rewards. But quickly after around 5 iterations, they managed to improve the reward to above 0. The policy curve moves faster to be near 0. The figure 6 average steps reflected the same thing, the two algorithms wasted a lot of steps at first, then figured out the way and got a high reward within fewer steps.

Overall, for the small MDP problem, the policy iteration performs better than value iteration, it converges faster and achieved a high reward. Even though it takes a slightly longer computing time for each iteration, it's worth it.

The result of value iteration is shown in figure 7 below. It took 102 iterations to converge to an optimal policy. As the image shows, the π(s) lets the agent move clockwise to be away from the -100 trap, and move closer to the +100 reward. It's interesting that when the agent probabilistically falls into the right notch area(circled in blue), it let the agent pick 3 directions towards the wall to avoid the possible falling into the -100 trap. Because the design of the grid is, if the agent can't go in the directions suggested, it has to go in the direction that will proceed to a new state, the agent has to move to the right at this state. Also, it's interesting to note that if the agent is on -100 trap, it will go up to get +100 reward instead of going through the long route with -1 reward on each step.

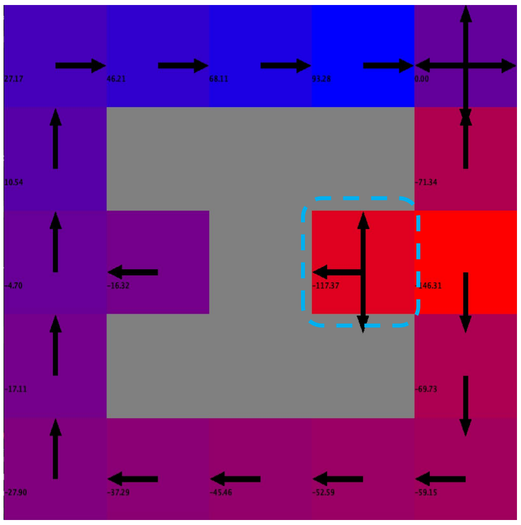

Figure 7. Value Iteration Policy

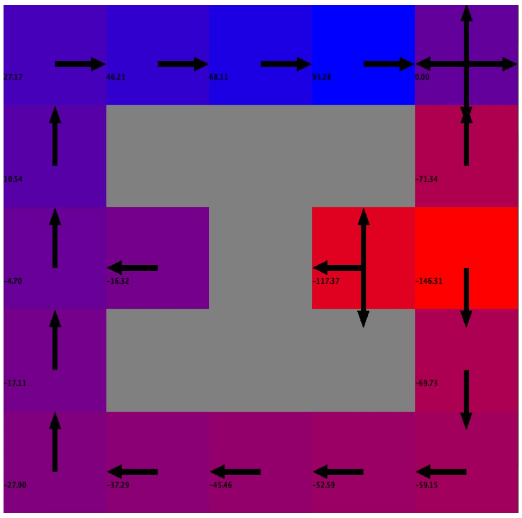

Figure 8. Policy Iteration Policy

Then I ran policy iteration on the small MDP. And it took 49 iterations to converge to an optimal policy, the map is shown in figure 8. As we can see, the policy also recommends a clockwise movement to get the 100 reward, and keep away from the -100 square. We can observe that the policy iteration produced the same policy as the value iteration, but the policy iteration converges much faster.

Q-learning (Convergence criteria 0.25)

To solve the small MDP by Q-learning, I first tested how different initial conditions will influence the convergence speed, I tested 3 different Q0, -100, 0, 100. The learning rate is set to be 0.5, the discount factor is 0.9, and Epsilon is 0.5. I got the result shown in figure 9. We can observe that the -100 and 100 initial condition curves are less smooth than the 0.0 initial condition curve. The 0.0 initial condition converges the fastest, comparatively, it's the best initial condition for the small MDP.

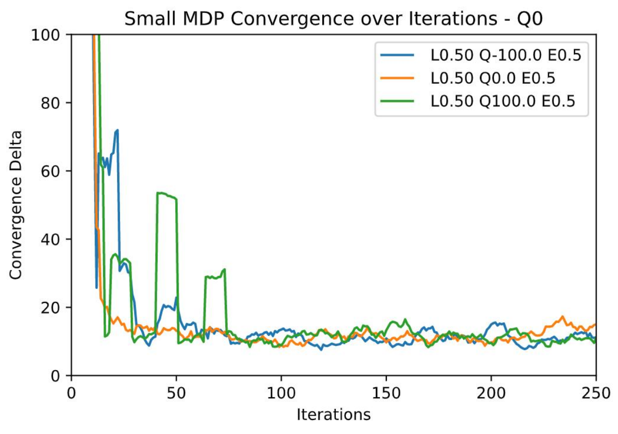

Figure 9

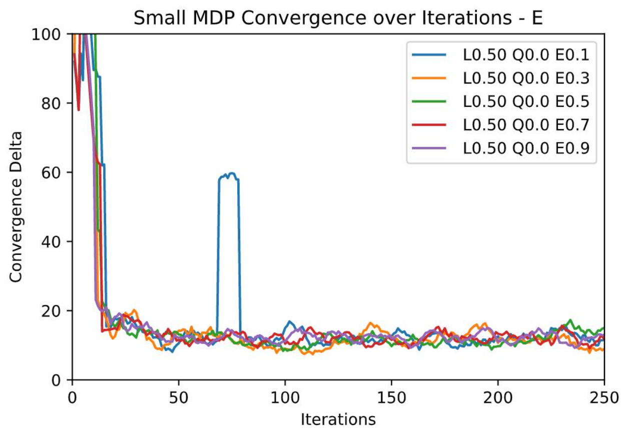

Figure 10

Then I test the epsilon value at 0.1, 0.3, 0.5, 0.7, and 0.9 and get the figure 10 above. The epsilon value change didn't largely influence the convergence value, so I test again with the learning rate combined. The plots for convergence and reward are shown in figure 11 and figure 12. From the chart, we can see the learning rate of 0.01 and epsilon of 0.3 converges very fast, but the improvement of reward is slower than the learning rate of 0.1. To explore the learning rate more thoroughly, I tested the learning rate only in figure 13. From the learning rate chart, we can be sure that the 0.01 learning rate converged the fastest.

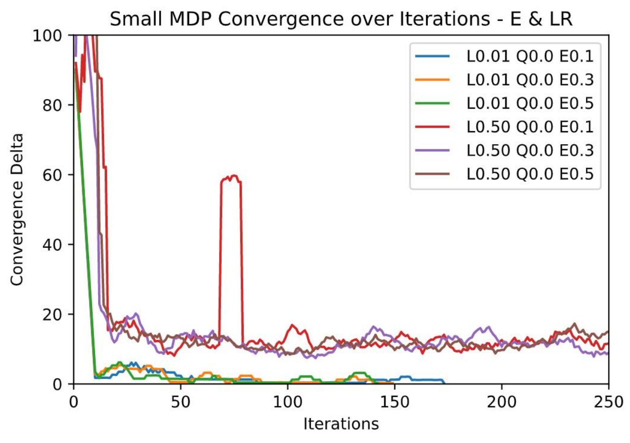

Figure 11

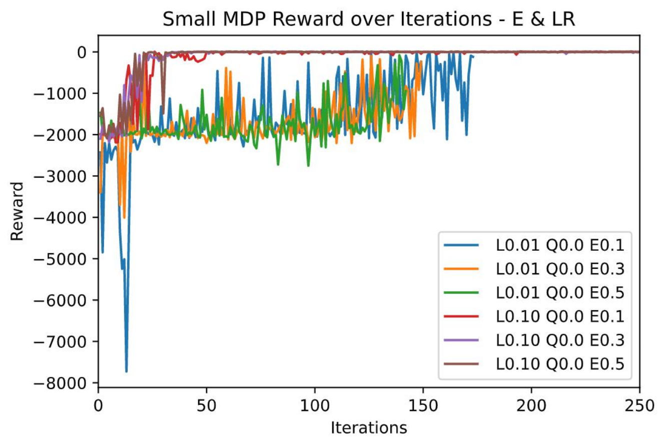

Figure 12

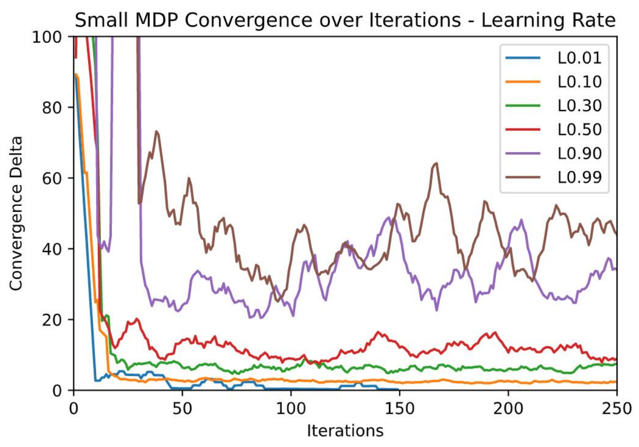

Figure 13

Then I ran Q-learning on the small MDP with a 0.01 learning rate, 0 initial condition, and 0.3 epsilon, the resulting map is shown in figure 14. After 1176 iterations it converges to an optimal policy. As we may expect, the policy also show clockwise movement, which is the same as the results of value iteration and policy iteration. The only difference is for the terminal state reward, it's -100 while in value iteration and policy iteration it's 0, but this won't influence the policy function(no matter the actions the agent needs to get final state to terminate).

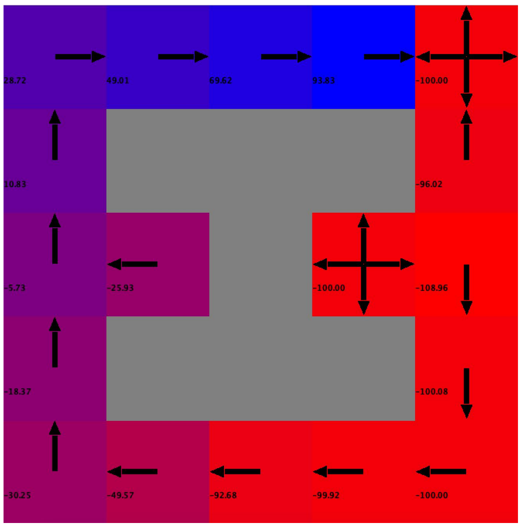

Figure 14. Q-learning Policy

In figure 15 and figure 16 I compared the Q-learning convergence speed and computing time with value iteration and policy iteration, the Q-learning is taking more iterations to converge because the Q-learning is a model-free algorithm and it takes more steps to explore and exploit the environment. As for computing time, Q-learning takes less time for each iteration.

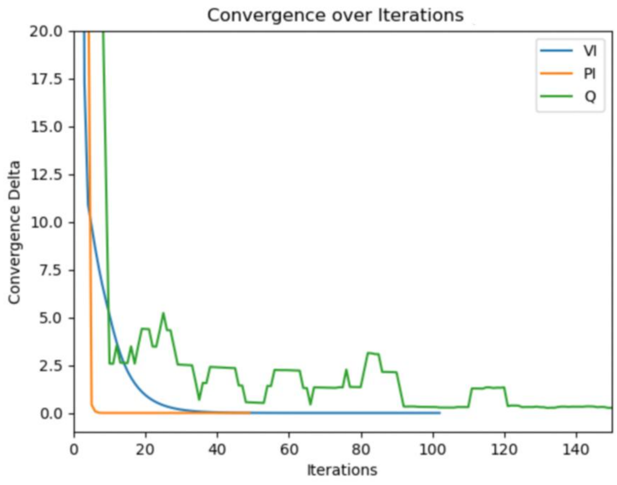

Figure 15

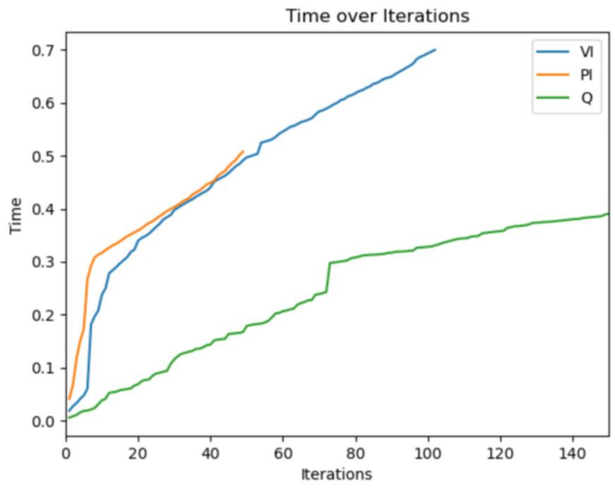

Figure 16

5. Large MDP

Value Iteration & Policy Iteration (Convergence criteria 10^-6)

The chart below(figure 17) is the convergence of value iteration and policy iteration. Same as for the small MDP, I ran 100 times for the 2 algorithms, each algorithm will stop iterating once reached the convergence criteria. For value iteration and policy iteration, the convergence criteria is that the convergence delta < 10^-6. Observe from the charts, the policy iteration converges faster than the value iteration, which is the same as in the small MDP. The policy iteration curves drop quickly and get near 0 within 10 iterations. And the value iteration is a more smooth curve, it gets near 0 around 50 iterations. For computing time in figure 18, the policy iteration takes longer than value iteration in each iteration.

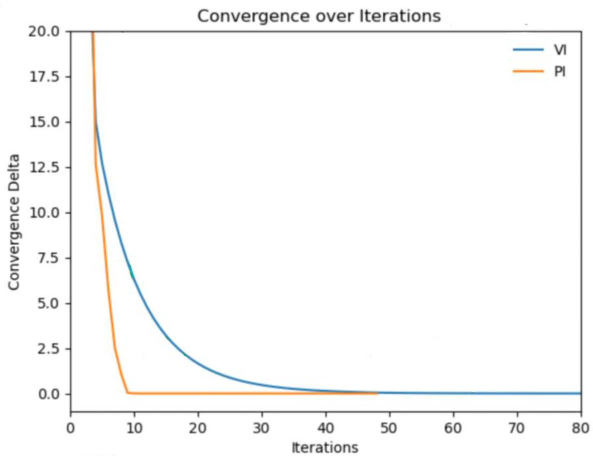

Figure 17

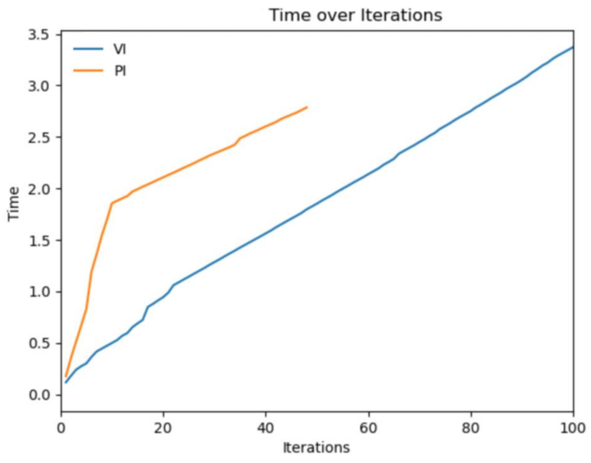

Figure 18

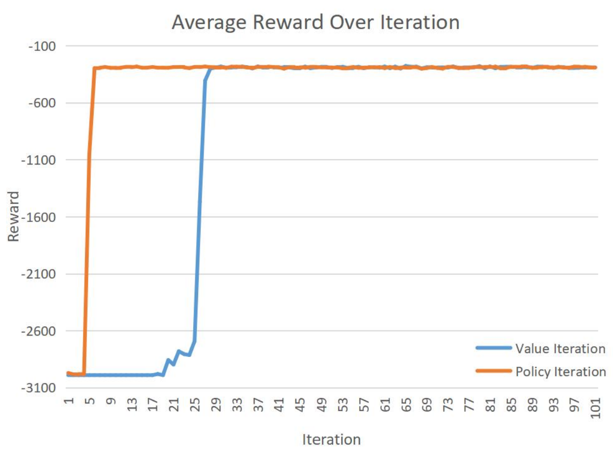

Figure 19

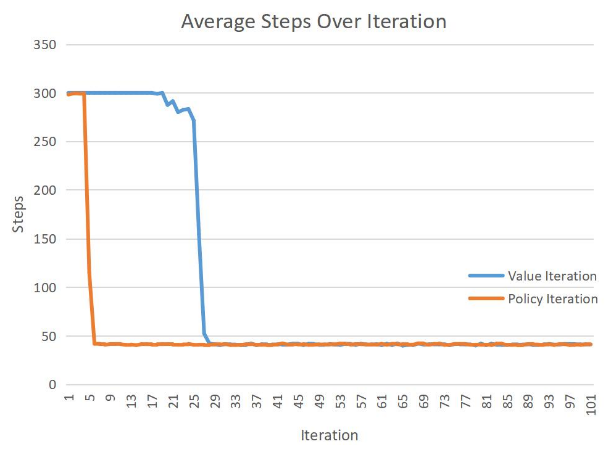

Figure 20

Figure 19 shows the average reward of value iteration and policy iteration. Observe from the chart, both algorithms wondered in the grid world at the first and got a low reward. The policy iteration reward improved after 5 iterations, combined with the average steps in Figure 20, the policy iteration found the path to the terminate states and started to improve the performance. The value iteration shows the same behavior but is slower than the policy iteration.

For the large MDP problem, the policy iteration also performs better than value iteration. Compared to the small MDP, more states in the MDP enlarge the performance gap between policy iteration and value iteration.

After running the value iteration, the policy map is shown in figure 21. It took 137 iterations to converge to an optimal policy. As the image shows, the π(s) lets the agent move towards the correct path, and try as much as it can to avoid going to the wrong rooms where the trap exists.

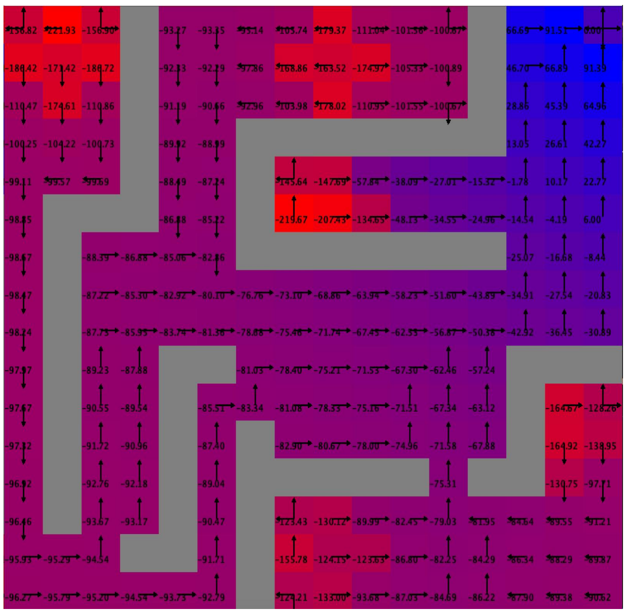
Figure 21. Value Iteration Policy

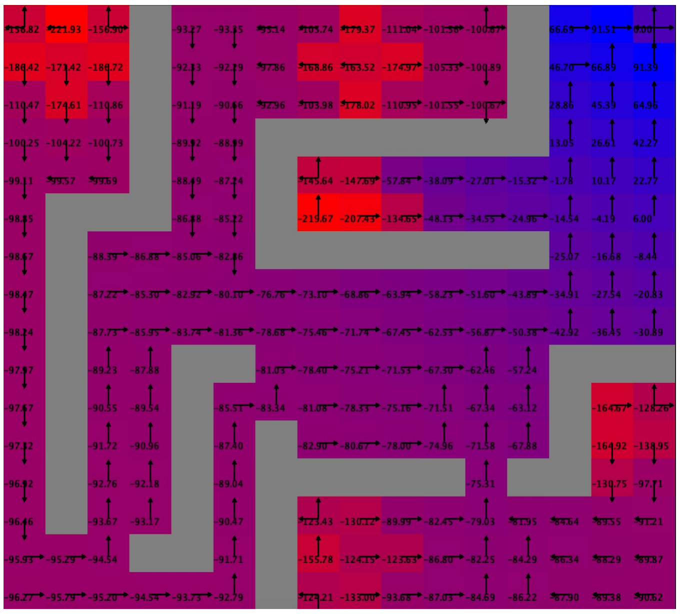
Figure 22. Policy Iteration Policy

The policy iteration map is shown in figure 22, the policy iteration took 48 iterations to converge to an optimal policy. The policy is the same as the value iteration, which tries to get the agent on the right path to the 100 reward and keep away from the wrong rooms and traps.

Q-learning (Convergence criteria 0.25)

To find the best parameter to solve the large MDP by Q-learning, I tested the following parameters: initial condition, learning rate, and epsilon. First I set the learning rate and epsilon to be 0.5 and plotted the convergence curve of Q = -100, Q = 0, Q = 100. The result is shown in figure 23. We can see that the convergence of Q = -100 dropped the fastest, but then raised up again, the final states for all three curves within 500 iterations are similar. So I chose the initial condition Q0 to be -100, and studied different learning rates and epsilon combinations, the result is shown in figure 24. As the plot shows, the learning rate of 0.01 and epsilon of 0.3 converged the fastest. Overall the learning rate of 0.01 performed better than the 0.5 learning rate.

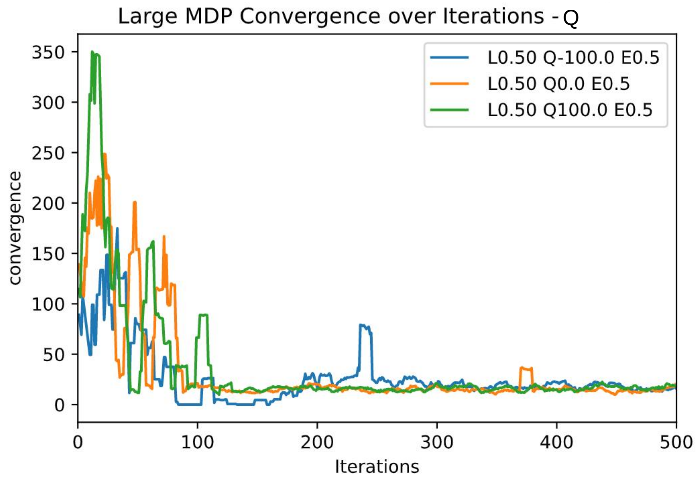

Figure 23

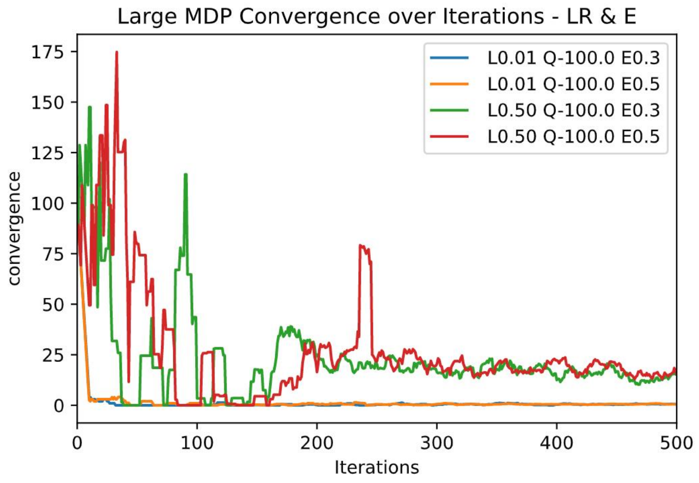

Figure 24

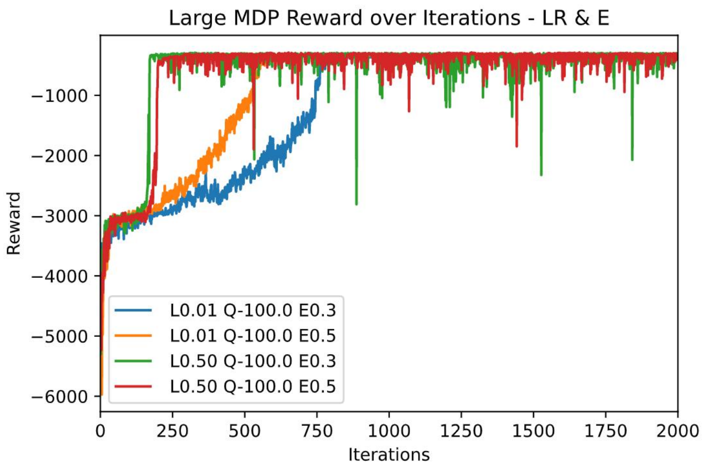

Figure 25

I also plotted the reward curves for these parameters in figure 25. The 0.5 learning rate reward improved faster than the 0.01 learning rate, but the final states of all curves appear the same.

Then I ran Q-learning on the large MDP with a 0.01 learning rate, -100 initial condition, and 0.3 epsilon, the resulting map is shown in figure 26. The Q-learning converges to an optimal policy after 1701 times iteration. The policy looks similar to the policy got from value iteration and policy iteration, but there are some different states. The policy is trying to avoid going to the wrong room but it’s not as decisive as the value iteration and policy iteration.

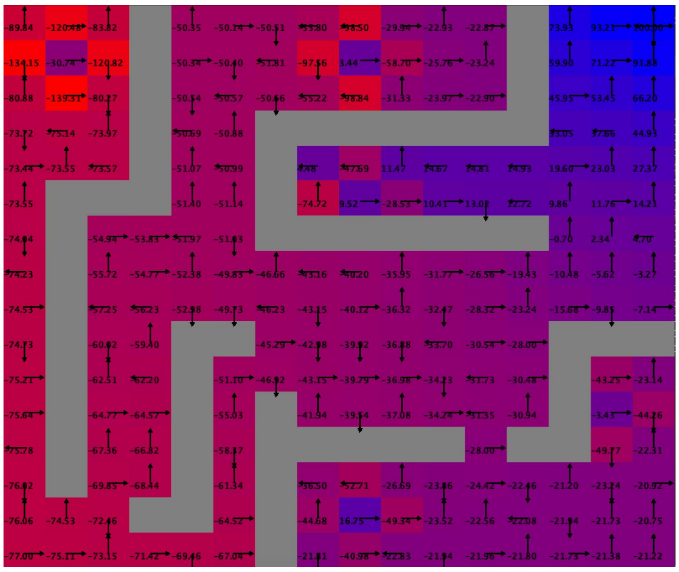
Figure 26. Q-learning Policy

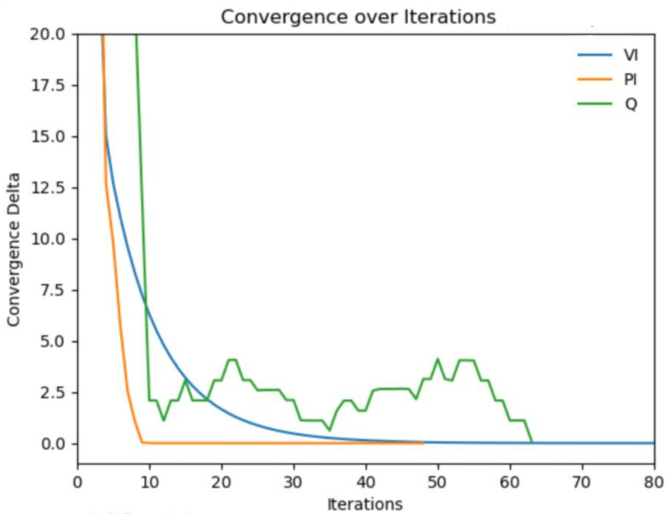

Figure 27

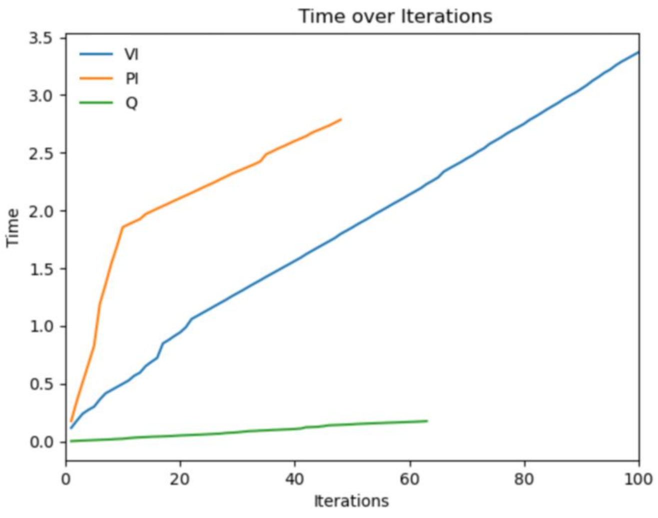

Figure 28

In Figures 27 and 28 I compared the convergence and computing time with value iteration and policy iteration. We can observe that the Q learning convergence is slower than the other two, and computing time is faster than value iteration and policy iteration. This observation is the same as the small MDP.

6. Conclusion

In the two MDP, I used three algorithms to find the optimal paths. For the value iteration and policy iteration, the policy iteration converges faster than value iteration, the computing time of policy iteration in each iteration also longer than value iteration. The optimal policy maps of both algorithm are the same in my MDP. For Q-learning, the best learning rate for both MDP is 0.01, the best exploration rate is 0.9(ε-greedy), and the best epsilon is 0.3. For small MDP, the best initial condition is 0, and for large MDP the best initial condition is -100. In addition, Q-learning outperformed value iteration and policy iteration in computation time, while the convergence is slower than the other two algorithms.