CS7641 Assignment 4: Markov Decision Processes

Xiaoming Su

Introduction

This assignment aims to solve two grid world problems using Markov Decision Processes namely value iteration and policy iteration as well as one reinforcement learning algorithm. Experiments included parameter tuning, convergence analysis and performance comparison. Different algorithms were evaluated and compared.

1. Two MDPs and why they are interesting

Two grid world problems each has only one agent were designed for this assignment. In the grid world, an agent tries to go to the terminal state from the start state.

1.1 Small grid world

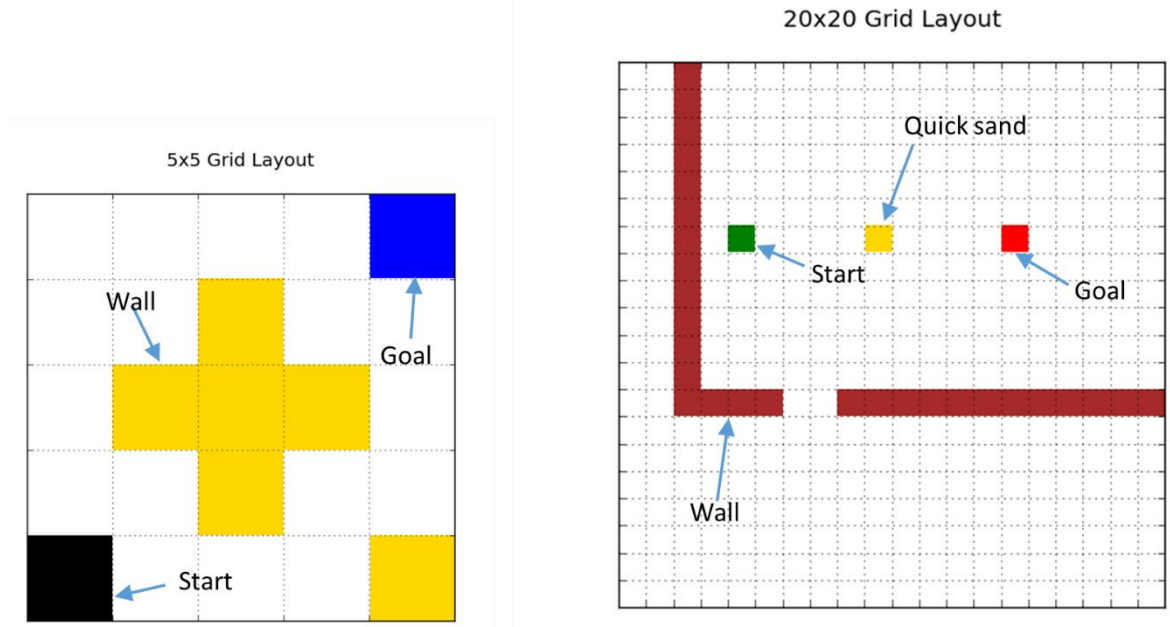
Small grid world is 5x5. The small grid world has fewer number of states and the problem is relatively simple. The grid world consists of one start, one terminal state and walls. The layout of small grid world is shown below. For this problem, a negative reward (-1) was assigned to all states except the terminal state. The reward of the goal was set to 100.

This problem is interesting because it has a simple but not trivial. The policy can be easily explained and understood. Thus it is a good choice for the study of convergence, running time and effects of parameters.

1.2 Large grid world

Large grid world has 20x20 grid. The grid world consists of one start, one goal, wall and quick sand. The terminal state has a reward of 100, quick sand has a negative reward -1000 and all the other state has a reward of -1. The quick sand has a converge possibility of 0.25.

The agent will go from the start to the goal. Instead of going in a straight line, it will try to avoid the quick sand and take a detour. The large grid world has relatively large number of states and has more complicated policies. Thus it is ideal to compare the performance of different algorithms.



(a) Small grid world (b) Large grid world

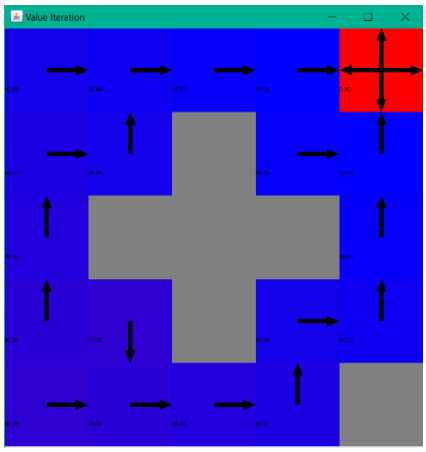
Figure 1. Layout of grid worlds (a) Small grid world; (b) Large grid world

2. Small grid world

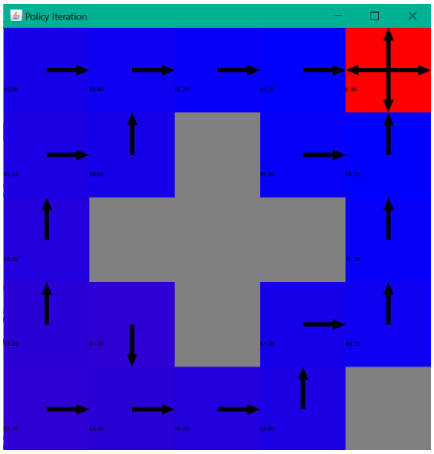
The small grid world problem was solved by value iteration, policy iteration and Q-learning using Burlap in Java. The transition model has 0.8 probability moving in the direction of targeted action, and the other 0.2 equally shared by the rest three directions.

2.1 Generated Policy

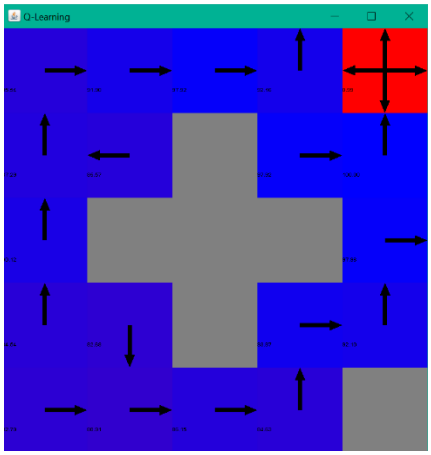
Q-learning used epsilon-greedy policy with epsilon = 0.1, discount factor = 0.99, initial Q-value to user everywhere = 0.99, learning rate = 0.99. The learning rate was set to be very fast. The resolved policy using different methods are shown in Figure 2.



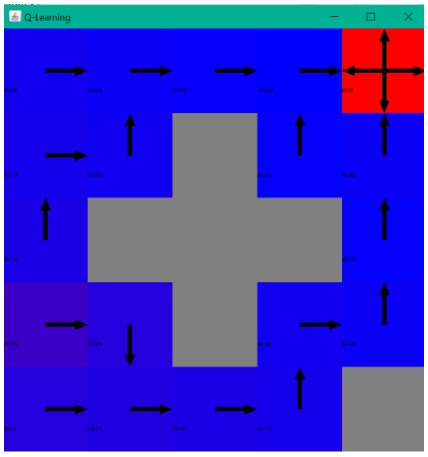
(a)Value Iteration



(b)Policy Iteration



(c)Q-learning_100



(d)Q-learning_150

Figure 2. Policy Plots Max iteration = 100 for (a), (b), (c); (d) Max iteration = 150 for Q learning

When the max iteration was set as 100 for value iteration, policy iteration and Q-learning, policy and value iteration methods gave same policy. For the Q learning, the policy was not really good. The agent could not get to the goal easily. The agent was likely to collide onto the right boundary. When max iteration was increased to 150, the agent was able to get to the goal. However, the policy is still not optimal. When the agent goes up, the agent would go right and return to the previous route. The results indicated that Q-learning needs more iterations to converge for this problem. Given a relatively small number of iterations, the policy generated by Q-learning is less optimal than the policy generated by value or policy iteration.

In order to converge faster, I changed the discounted factor from 0.99 to 0.9, by letting it considered more about “current reward”, the max iteration was 100. It turned out that the lower discounted factor gave better policy (Figure 3) compared with Figure 2 (c).

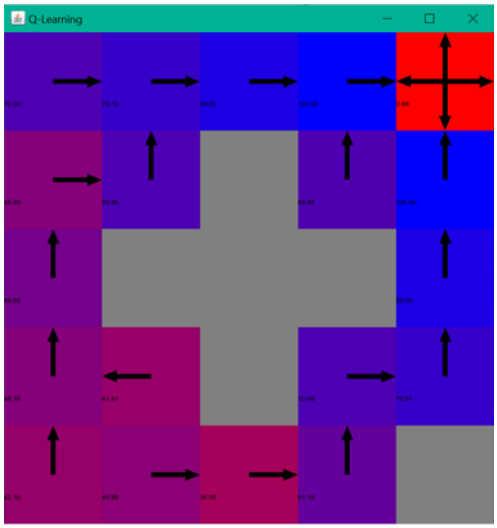


Figure 3. Policy given by Q-learning.
Max iteration = 100, discounted factor = 0.9.

2.2 Comparison and discussion of PI, VI and Q-learning

Three methods were compared on running time, rewards and number of steps to goal. The discounted factor was set to 0.99 for all three methods.

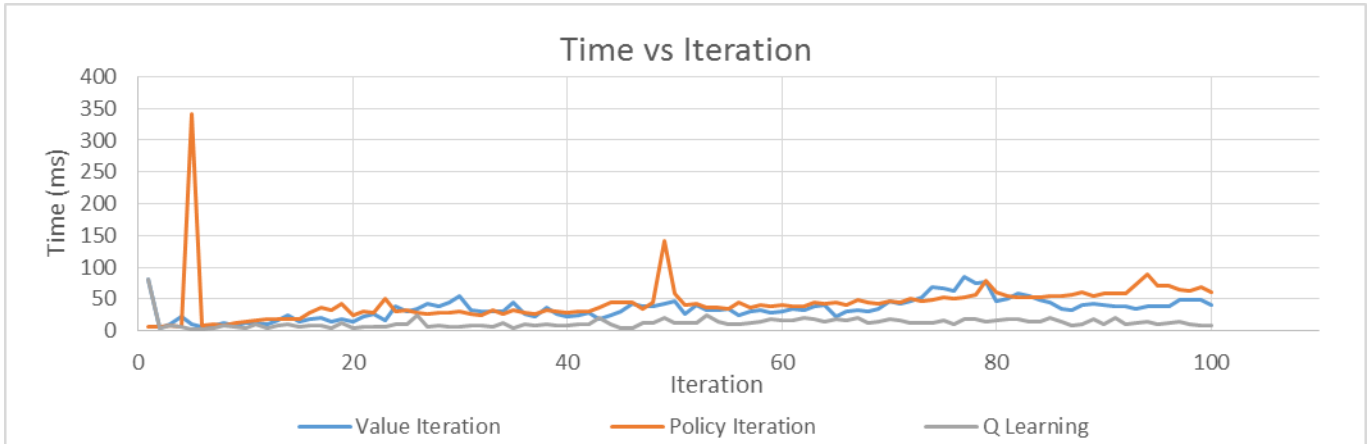


Figure 4. Time vs Number of iterations

For value iteration and policy iteration, it was clear that the running time grows linearly with the increase of iterations. Policy iteration takes slightly longer time than value iteration. For the PI, all possible state and action pairs from the transition probability P(s,a,s') based on the predetermined policy "pi" need to be calculated. VI used Bellman equation by evaluating progressively state-by-state until the solution converges. Thus PI is more computationally expensive by calculating the policy. For the Qlearning, running time grow very slow when the number of iterations are large, instead of conducting complex computation, it just hashes the actions taken and the rewards achieved.
Thus the running time ranks as follows given same number of iterations: PI > VI > Q-learning.

Figure 5 shows the number of steps to the goal for three methods given same number of iterations. VI and PI converged at about in about 3-5 iterations while Q-learning converged in about 12 iterations. When the number of iterations were very small (before convergence), the number of steps for PI was much less than that of VI. It makes sense because PI was more computationally expensive and provided policy directly.
Both number of steps for PI and VI when iterations were large didn’t change significantly. For the Q-learning,

even when the number of iterations was large, there was still spikes occasionally. Q-learning for this problem utilized epsilon-greedy policy with epsilon = 0.1. Epsilon greedy policy is a way of selecting random actions with uniform distribution from a set of available actions. Using this policy random action is select random action with epsilon probability. The occurrence of spikes was due to the randomness we introduced.

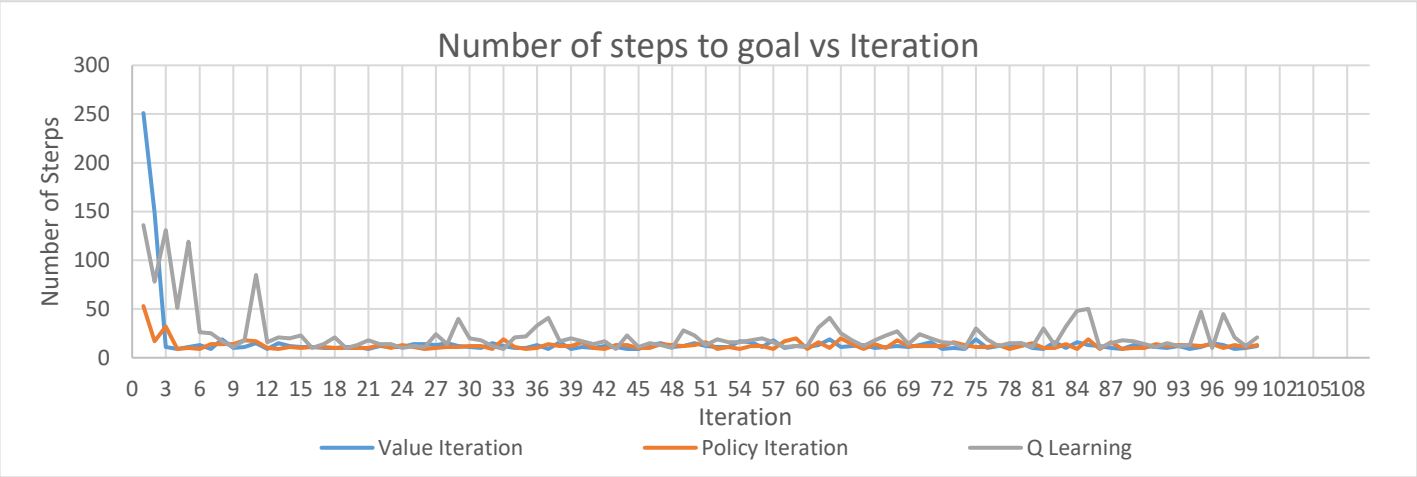


Figure 5. Number of steps to get to the goal vs Number of iterations

The reward for each state is -1 except the terminal state (100). For all the methods, when the number of iterations was, the rewards were all about 100 which indicated that the goal was reached. Similar to Figure 5, Figure 6 showed that PI and VI converged in about 3-5 iterations. Q-learning converged in about 12 iterations. The spikes of rewards for Q-learning corresponded to the spikes in steps.

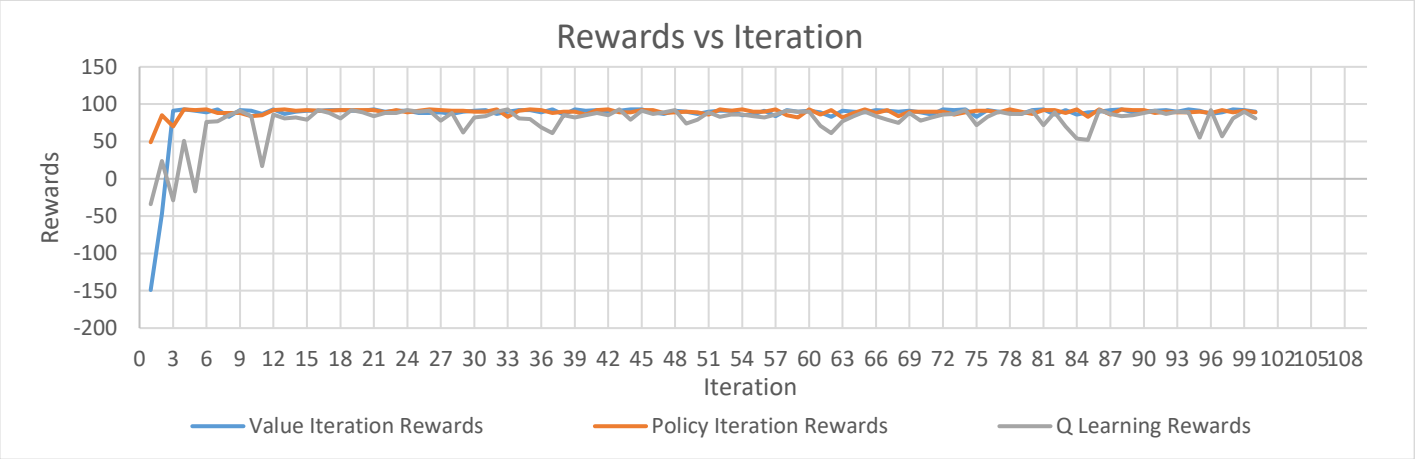


Figure 6. Rewards vs Iteration

Comparison:

Q-learning	Model-free method
PI	Model-based method, based on Bellman operator, linear.
VI	Model-based method, based on Optimality Bellman operator, non-linear

Bothe VI and PI can be seen as two cases of the generalized policy iteration.

3 Large grid world

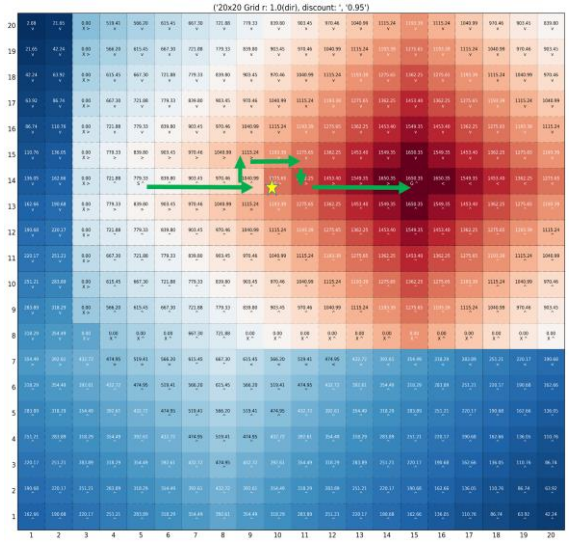
This grid world is much more complicated. The reward for the goal was 100, other state besides quicksand

was -1. Unlike small grid world, there was a quick sand in the world with large negative reward -1000 between a direct route from start to the terminal state. In order to get to the goal with high reward, the agent need to take a detour. Walls were set to be around the start. There was a hole on the wall which helped to study the propagation. This problem was implemented and solved by three methods using Mdptoolbox in Python.

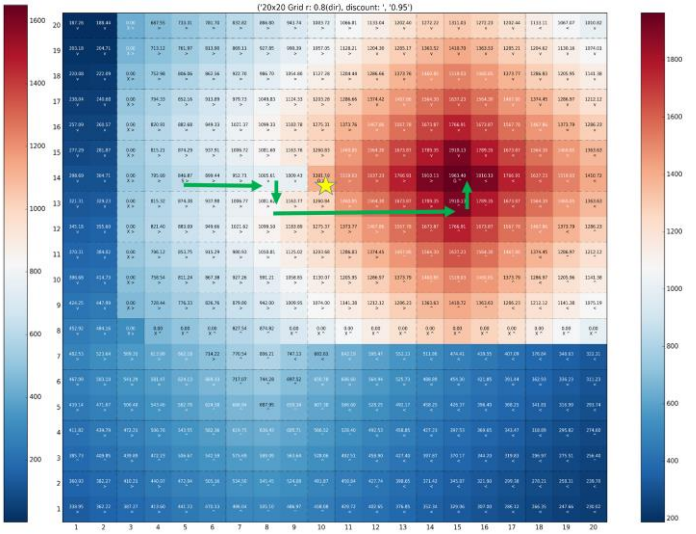
3.1 Value iteration

For the VI, experimental parameters were set as follows: discount = 0.95, epsilon = 0.00001, max_iter = 10000. The influence of action accuracy was studied using VI. Action accuracy k was set to 1.0, 0.8 and 0.7.

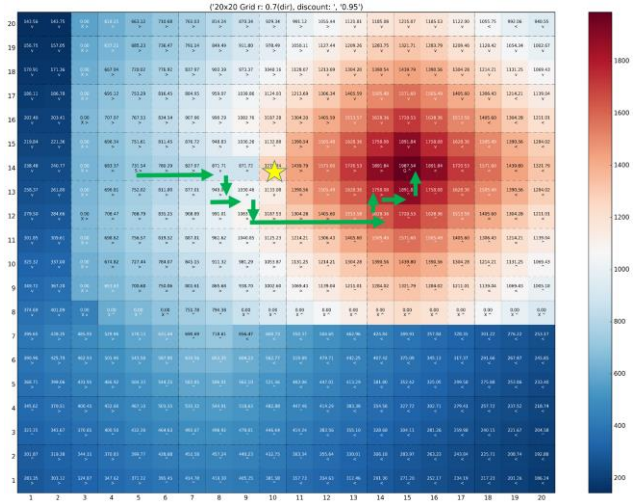
The generated policies were shown in Figure 7. The green arrows in the plots indicated the route from the start to the terminal state. The yellow star represented the quicksand. When k = 1, the action was always right. The agent just took a very small detour. Then k = 0.8, the detour was larger. When the action accuracy became even smaller 0.7, the detour was larger. It makes sense. When the action accuracy was lower, there was a larger possibility for the agent to get stuck in the quicksand when it was moving near the quicksand. Thus, when the action accuracy was lower, agent moved further away from the quicksand. Value bar was shown on the side, red means higher number, blue means lower number. The values outside the wall was very low, the values was propagated through the hole on the wall. The values of states around the terminal states was almost symmetrically distributed.



(a) k = 10



(b) k = 0.8



(c) k = 0.7

Figure 7. Policy generated by VI

3.2 Policy iteration

For the PI, experimental parameters were set as follows: k = 0.8, discount = 0.95, epsilon = 0.00001, max_iter = 10000.

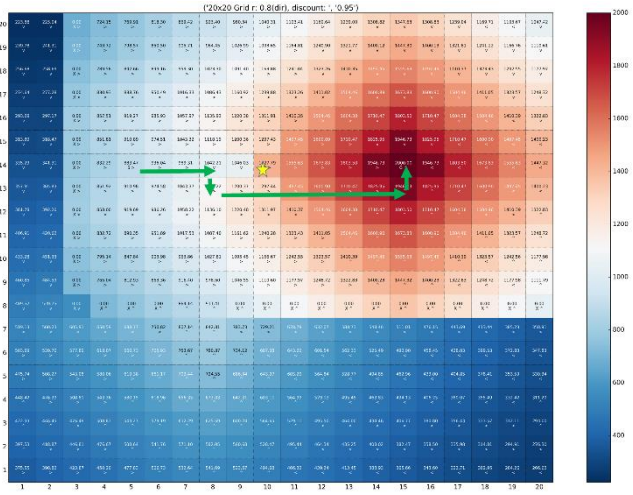


Figure 8. Policy generated by PI

By comparing Figure 8 and Figure 7 (b), we can find that the policies generated by PI and VI were pretty much alike. No matter in a small grid world or a large more complicated grid world, PI and VI usually generated very similar policies.

3.3 Influence of the number of states

	5x5 Time (s)	5x5 Iteration	20x20 Time (s)	20x20 Iteration
Policy Iteration (k=0.8)	0.007	4	0.145	10
Value Iteration (k=1.0)	0.024	372	0.023	34
Value Iteration (k=0.8)	0.023	372	0.040	78
Value Iteration (k=0.7)	0.025	372	0.134	99

Table 1. Comparison of time and number of iterations to converge

Table 1 shows the number of iterations and the amount of running time to converge for small grid world and large grid world using different methods. All the experiments performed for this table were conducted in Python using Mdptoolbox for consistence. All the parameters not listed on the table was set to be the same.

Policy iteration and value iteration converged to same answer. Previous experiments (policy plots) demonstrated this point. Policy iteration converges faster than value iteration items of running time and the number of iterations needed.

As the number of state becomes larger, the running time will be longer. It's hard to conclude how the number of iterations was influenced for these two grid worlds because they were different in other ways besides the number of states. For two grid worlds which are mainly different in the number of states, the number of iterations needed to converge will become larger.

3.4 Q-learning

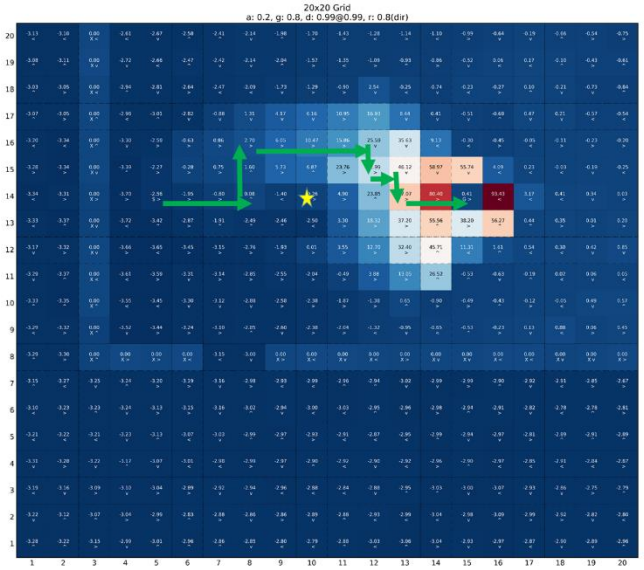
For Q-learning using epsilon-greedy policy, two sets of experiments were conducted. The experimental parameters were set as follows:

(a) k=0.8, learning rate alpha=0.2, discount rate gamma=0.8, rar=0.99 (epsilon = 0.01), rard=0.99, n_iter=1000000;

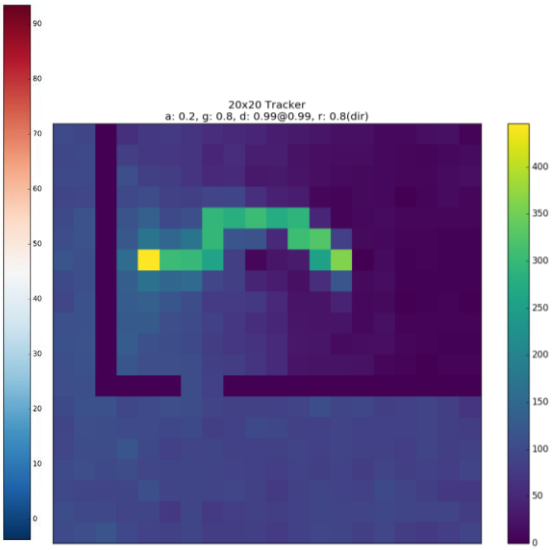
(b) k=0.8, learning rate alpha=0.2, discount rate gamma=0.8, rar=0.99 (epsilon = 0.01), rard=0.999999, n_iter=1000000.

Generated policies and corresponding tracker plots were shown in Figure 9. Brighter colors in the tracker plots indicated that grid is more explored.

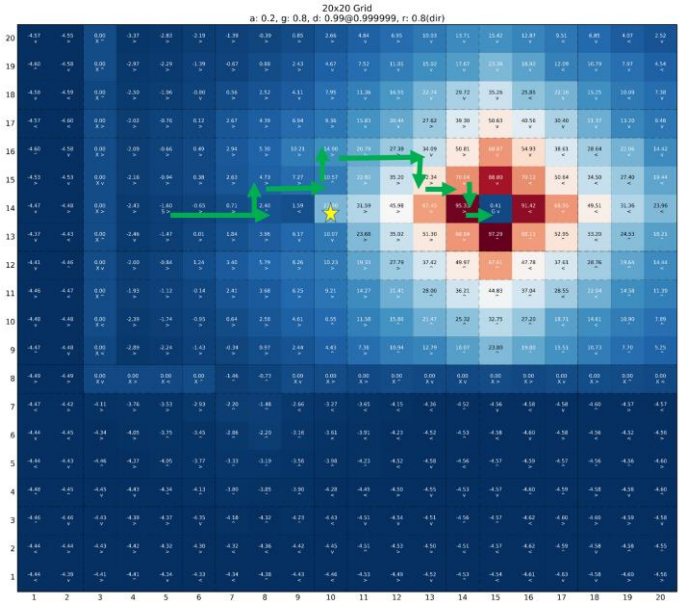
The major difference of the two experiments is the epsilon decay rate. The epsilon of experiment (a) decays much faster than that of experiment (b). As the randomness decreased much quickly in (b) during the iterations process, it explored less than experiment (a).



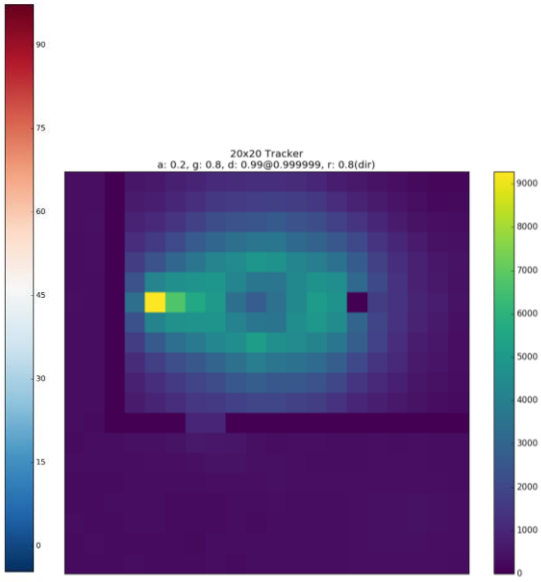
(a)Policy for Experiment a



(a') Tracker for Experiment a



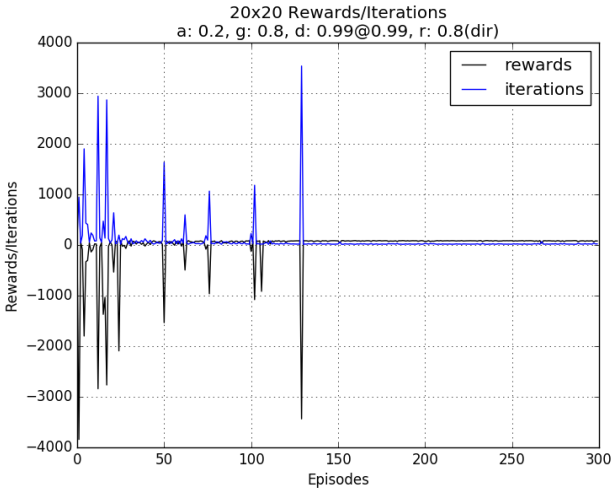
(b)Policy for Experiment b

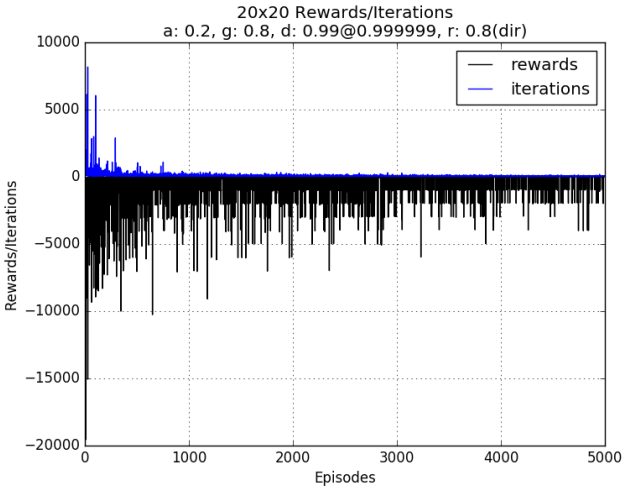


(b') Tracker for Experiment b

Figure 9. Policies generated by Q-learning and corresponding tracker plot

By comparing Figure 9 (a) and (b), we can find that in experiment (b) states were less explored than in experiment (a). When the agent had more freedom to explore, the values around the terminal state were more symmetrically distributed just like in PI and VI experiments. When epsilon decayed really fast, the agent would explore less and exploit more.

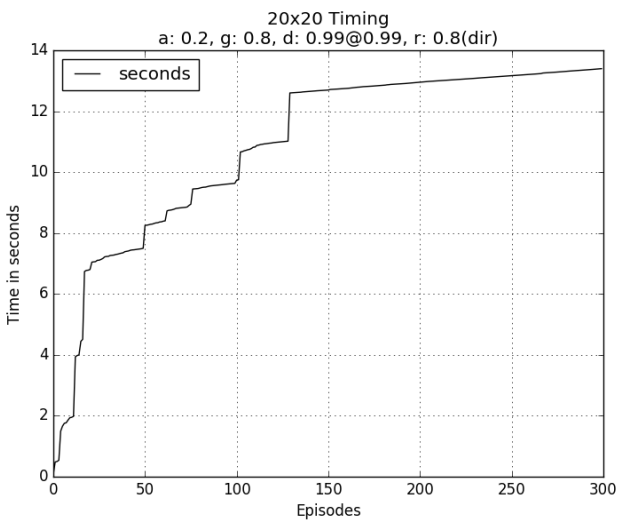


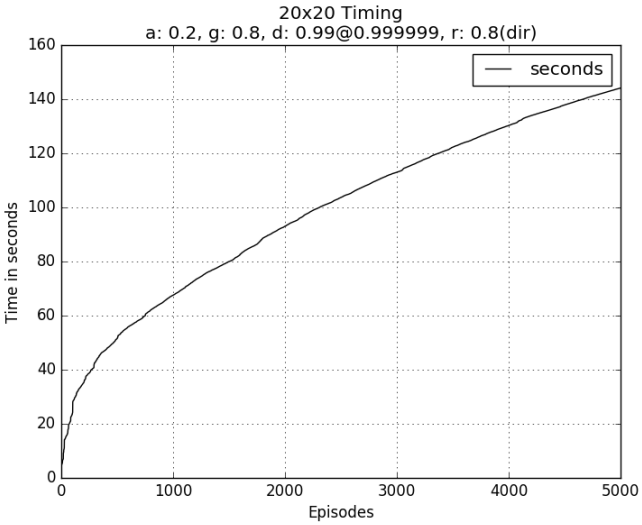


(a) Experiment a (b) Experiment b

Figure 10. Rewards/Iterations vs Episodes

Experiment (a) explored less and exploited more compared with experiment (b). From Figure 10, we can find that experiment (a) converged much faster than experiment (b). Because there was more randomness in experiment (b), we can find there were many spikes in plot Figure 10 (b). The spikes were still very obvious when the number of episodes was about 800, while spikes disappeared in (a) when episodes reached about 125.





(a) Experiment a (b) Experiment b

Figure 11. Time vs Episodes

Experiment (b) took longer time than experiment (a) given same number of episodes. It makes sense because it explored more.