CS7641 Spring 2022 Assignment 4

Jian-Yuan Lin

jasonlin0211@gatech.edu

1 INTRODUCTION

1.1 Problem Statement

In this assignment, two Markov Decision Process(MDP) example, the Frozen Lake Problem and the Forest Management Problem are presented. Three reinforcement learning methods including Policy Iteration, Value Iteration and Q-Learning are utilized to solve these problem for study and comparison purpose. The details and performance of each algorithm on the previous two problem will be analyzed in this assignment.

1.2 Frozen Lake Problem

Frozen Lake is a discrete MDP problem like grid world[1]. In this problem, a 4 * 4 grid is created with four components: S (starting cell), F (frozen cell), H (hole) and G (goal). An agent is initially placed on S, which is on the top left corner of the grid. The agent can go up, down, right and left, and the agent's goal is to reach G, which is placed in the bottom left corner of the grid. F is the safe places to pass, while H is a game over stop and should be avoided. Since the frozen floor is slippery, the movement accuracy will be only 33 %, while 33 % to the left side and right side of the agent, respectively. If the agent reaches G, it will get reward +1, and for the other grid it will get reward 0. Figure 1 shows the set up of the problem. This is very similar to the robot navigation problem in real world, and besides the original 4 * 4 version, I have proposed 20 * 20 map to investigate the algorithm performance on different size of the problem.

Figure 1— 4 * 4 Frozen Lake Grid

1.3 Forest Management Problem

Forest Management Problem is another MDP problem [2]. A forest can be defined modeled as a discrete states : {0, 1 . . . S-1 }, with S-1 being the oldest state. A forest is managed by two actions: 'Wait' and 'Cut', and the objective for first action is to maintain an old forest for wildlife and the objective to second action is to make money by selling cut wood. Each year there is a probability p that a fire burns the forest, and if a forest is burnt, it will return to state 0. As the forest is in its oldest state, the reward for 'Wait' and 'Cut' is r1(4) and r2(2), respectively. This problem is very similar to the real world problem, since government needs to maintain a good balance between economy and nature, and some random natural disasters are also possible to happen.

2 ALGORITHMS IMPLEMENTATION

2.1 Reinforcement Learning Algorithm introduction

2.1.1 Policy Iteration (PI) and Value Iteration (VI)

Since Policy Iteration (PI) and Value Iteration (VI) are very similar approaches and therefore I combined the introduction for the comparison purpose. PI is composed by three steps [3]. First, randomly generate initial policies for each state. Step 2 is the policy evaluation, which gets the action for every state in the policy and evaluate the value using Bellman equation. Step 3 is policy improvement, which obtains the best action from value function for every state, and if the best action is better then the current policy action, just update the current action by the best action. Iterate step 2 and 3, until the policy doesn't change, then we can consider the algorithm is converged.

Different from PI, VI combines the policy evaluation and policy improvement steps by using Bellman equation as a update function to find the optimal value iteratively [4]. The algorithm converges when the values between new state and old state is smaller than a threshold. Once the optimal value function is found, the optimal policy can be created from the state values.

2.1.2 Q-Learning

Q-Learning (QL) is a model-free method, which does not require a model of the environment, and it finds an optimal policy by maximizing the expected value

of the total reward over any and all successive steps from the current states. First, a Q-Table with shape [state, action] is created to store the Q-Values, which are initialized as 0. In the following iterations, the agent can either use the Q-Table as a reference and view all possible actions for a given state, which is called "Exploit", act randomly to update the Q-Values, which is called "Explore". The termination condition of the algorithm could be set as max iterations or the threshold of improvement.

2.2 Frozen Lake Implementation

First, the PI and VI are implemented on the original 4 * 4 frozen lake grid and investigate the performance. For these two algorithms, the parameter that we can investigate is γ , which is the discount factor. This factor is a reflection of how you value your future reward. If the gamma is close to 0, that means the agent is very greedy about the current reward and less serious about the future rewards. In contrast, if the gamma is close to 1, that means the agent values the current and future rewards almost equally, and therefore will plan the policy according to the potential future states. The implementations results for PI are shown in the Figure 2. To investigate the discount factor effect, ten values ([0.05, 0.15, 0.25, 0.35, 0.45, 0.55, 0.65, 0.75, 0.85, 0.95]) are implemented, and the execution times, average rewards and the iterations to converge are presented.

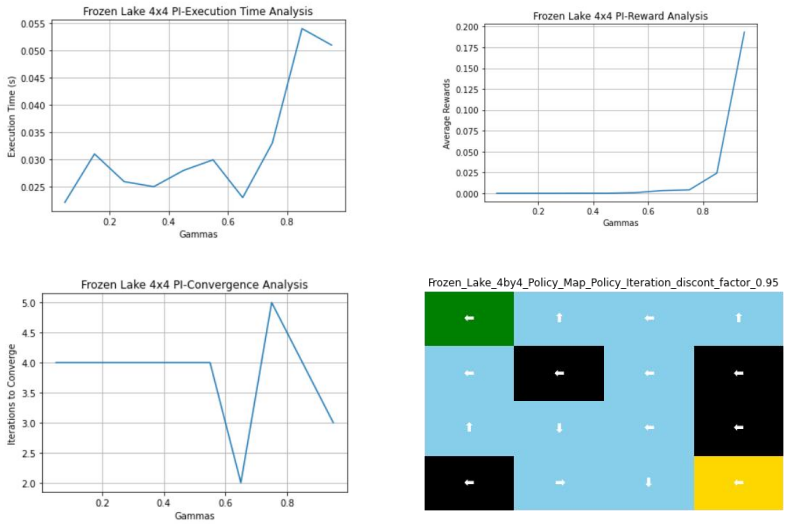


Figure 2—4 * 4 Frozen Lake Grid (a) PI runtime plot (b) PI average rewards plot (c) PI convergence plot (d) PI with discount factor = 0.95 Policy Map

It can be seen that the runtime increases as the discount factor increases, which makes sense since more information about the future rewards needs to be considered. The average rewards also increase with the discount factor, which indicates for this kind of stochastic problem, greedy approach won't work well due to the actual action might not be the exact action chosen by the policy. The converge condition I choose is the new policy is identical to the current policy, and PI actually converges in only few iterations. The policy map generated by choosing discount factor = 0.95 is also presented to see how the agent makes decision.

To compare with PI, VI is the next algorithm to be implemented. The convergence criteria is when the change of the summation of the values for every states are below a small tolerance (1e-20 in my algorithm). Same discount factors are implemented, and the results are shown in Figure 3. It can be observed that the runtime in VI is larger than PI, and the reason is the iterations required for convergence in VI is significantly more than PI, as shown in convergence plot. The average rewards are similar, and for the same discount factor (0.95), PI and VI converge to the same policy map, which means they converge to the same optimal result in this question.

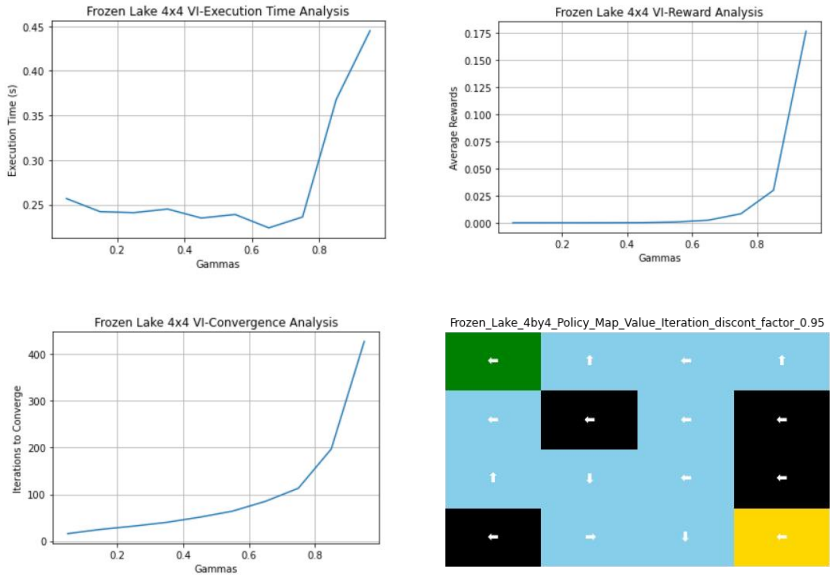
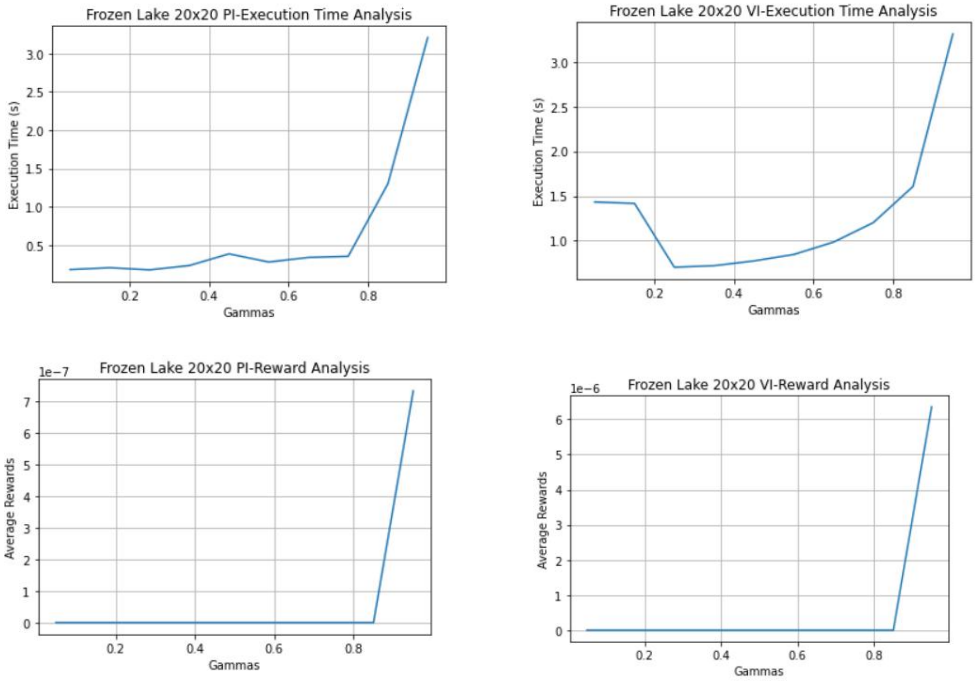


Figure 3—4 * 4 Frozen Lake Grid (a) VI runtime plot (b) VI average rewards plot (c) VI convergence plot (d) VI with discount factor = 0.95 policy map

After the investigation on the small scale problem, it will be interesting to see the performance of both algorithm on 20 * 20 grid. Figure 4 shows the performance comparison of PI and VI. It can be seen that similar to the previous experiments, the iterations required for VI is significantly larger for PI, but the runtime for VI is close to PI in this experiment. Therefore, we can conclude that both algorithms are guarantee to converge, and PI requires less iteration to converge [5], but each iteration for PI is more complicated than VI, so which algorithm is faster depends on the environment. VI performs better on average rewards, but since the rewards are all very small, we know that it is very difficult for the agent to actually reach the goal in 20 * 20 grid. In conclusion, in this frozen lake problem, PI and VI can achieve similar performance on rewards, but PI tends to finish faster in most scenarios, so it will be recommend to implement PI then VI since we could validate the result faster in PI.



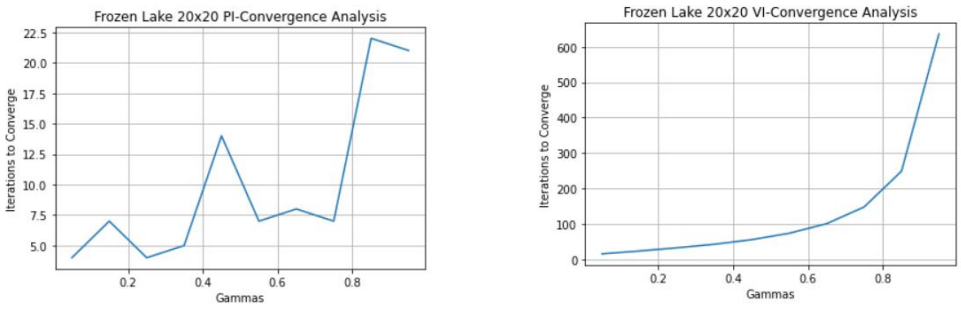


Figure 4—20 * 20 Frozen Lake Grid PI and VI (a) runtime plot (b) average rewards plot (c) convergence plot

For QL, I choose to use the gamma = 0.95 and the maximum iteration = 30000 as previous experiment shows that higher discount factor tends to perform better. The parameters that I would like to investigate is the epsilon, which is the threshold for the agent to choose either to "Explore" or to "Exploit". For every iteration, the agent will randomly generate a number between 0 - 1, and if the this number is smaller than epsilon, the agent will act randomly to explore different action in the action space, but if this number is bigger than epsilon, then it will follow the previous result in Q-Table and choose the action accordingly. Therefore, if epsilon is close to 0, the agent tends to exploit more rather than explore, and if the epsilon is close to 1, the agent tends to explore more. This is so-called Epsilon-Greedy Q-Learning [6]. I choose to test epsilon = [0.01, 0.05, 0.1, 0.2, 0.3, 0.4] on the 4 * 4 frozen lake grid, and the experiment result is presented in Figure 5.

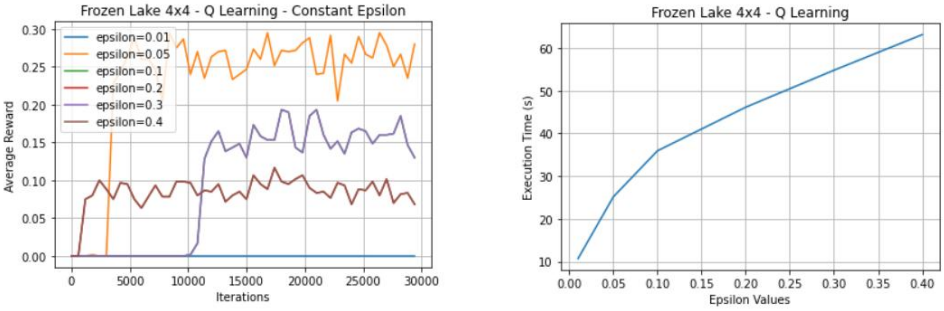


Figure 5—4 * 4 Frozen Lake QL (a) Average Reward plot (b) Runtime plot

Compare to the previous two algorithms, the best scenario in QL(epsilon = 0.05) can achieve even better rewards, but since the exploration contains randomness, it is not guaranteed that every time QL will be better than PI and VI. However,

we can observe that if epsilon is too small, it's performance is bad (rewards = 0 means not being able to reach the end). But if the epsilon is increased too much, the performance will downgrade since too much exploration will actually mess up the learned policy in Q-Table. In terms of the runtime, apparently PI and VI are on the advantage side. We can observe that the runtime increases when the epsilon increase, and that's because explore takes more time than exploit since exploit just need to reuse the previous results, and higher epsilon leads to more exploration. On the larger scale of the 20 * 20 grid, QL performs worse than the PI and VI. All of the QL with different value of epsilon can't reach the goal, and that might due to the random exploration didn't reach the goal, and if I increase the iterations limit, the result might be better, but the runtime will also increase. Therefore, it can be concluded that compared to PI and VI, QL is not very applicable to the larger scale problem, since the runtime and computation resource will be significantly larger than PI and VI.

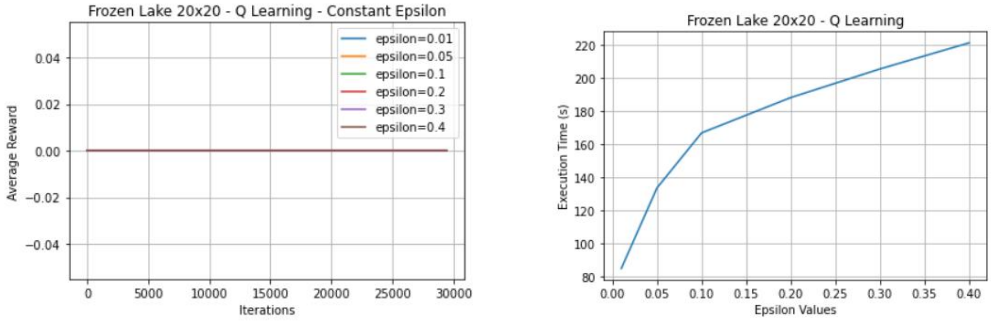
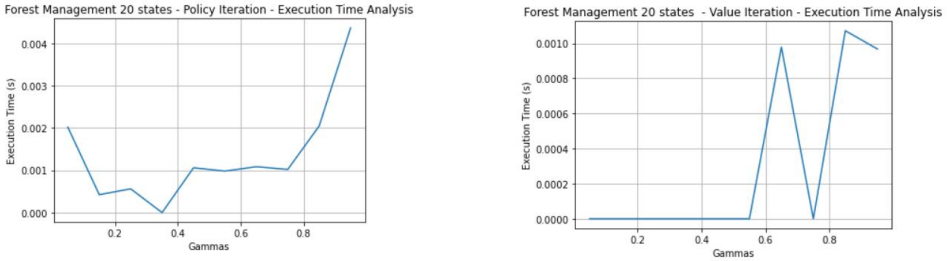


Figure 6—20 * 20 Frozen Lake QL (a) Average Reward plot (b) Runtime plot

2.3 Forest Management Implementation

For forest management problem, I conducted the same study for the discount factor in PI and VI, and the states number is set as 20. The results for PI and VI are presented in Figure 7.



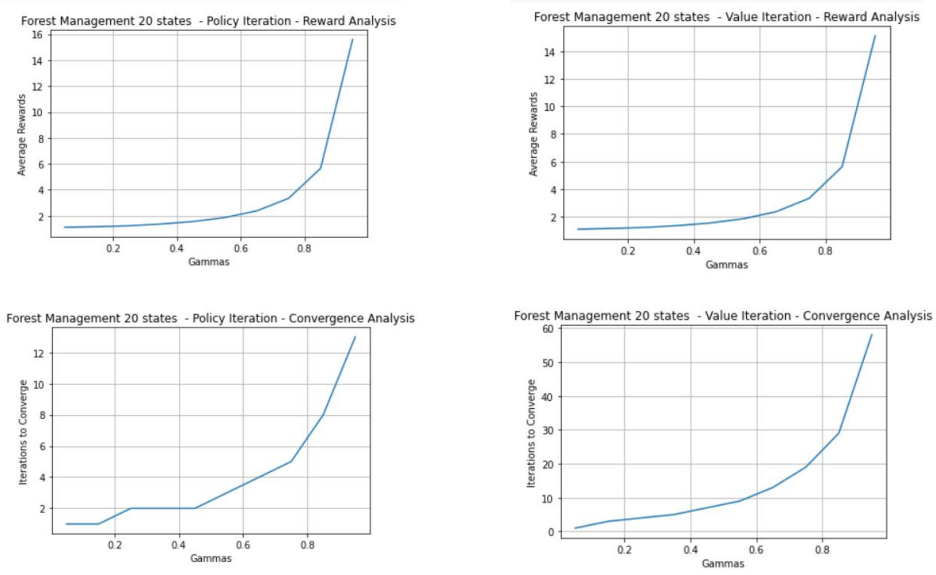
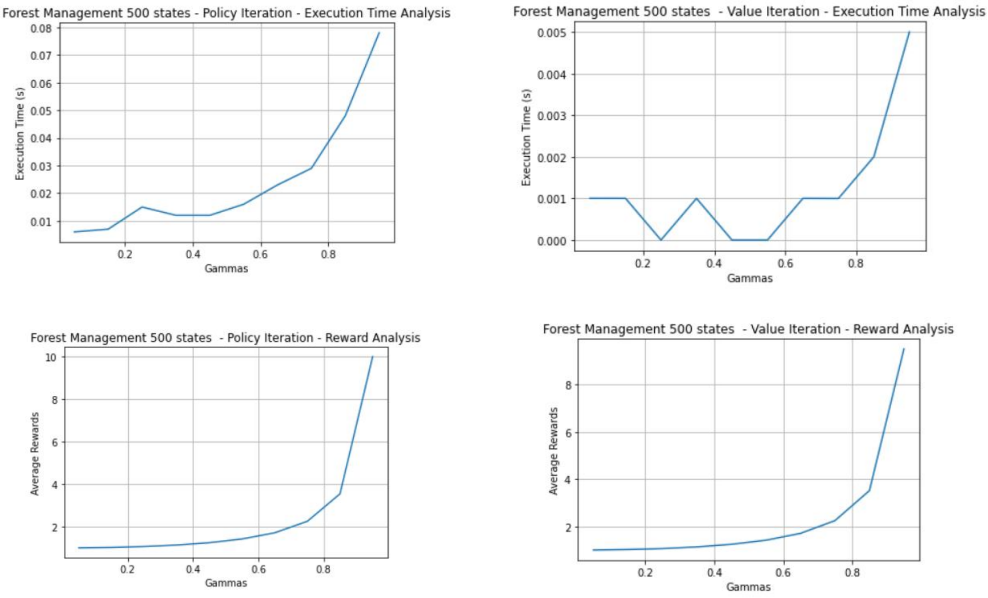


Figure 7—20 states Forest Management PI and VI (a) runtime plot (b) average rewards plot (c) convergence plot

The runtime for PI and VI are very close, but in terms of the iterations required to converge, VI still requires more iterations, so the experiment results here aligns with the previous conclusion. The average rewards are also almost identical, which means PI and VI will both converge to the optimal policy. To see how these two algorithms perform on a larger scale problem, I change the number of states to 500, and the results are presented in Figure 8.



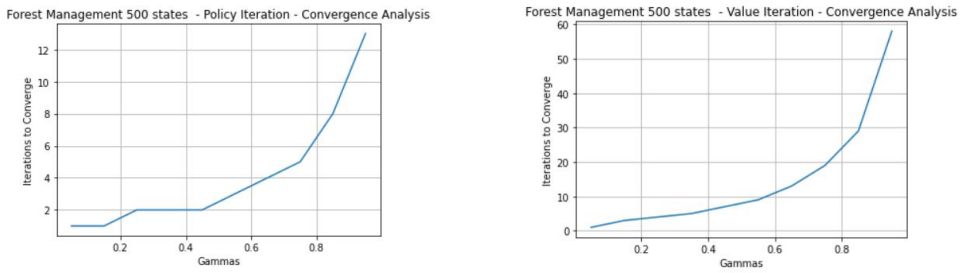


Figure 8—500 states Forest Management PI and VI (a) runtime plot (b) average rewards plot (c) convergence plot

The interesting part here is that VI actually has smaller runtime than PI, and this might be due to the scale of the problem increases, the every iteration for the policy iteration grows more, and therefore leads to the longer runtime. It should be noted that the result here still aligns with the previous conclusion, since PI still requires fewer iterations to converge. The average rewards are very close for PI and VI, which means they might converge to the similar optimal policy.

For QL, the same epsilon = [0.01, 0.05, 0.1, 0.2, 0.3, 0.4] is applied on the 20 states forest management problem, and the result is presented in Figure 9. The training runtime is significantly larger than PI and VI. The training results for forest management problem after many iterations with different epsilon in QL are quite similar, but in this problem they are all worse than PI and VI in terms of the average awards.

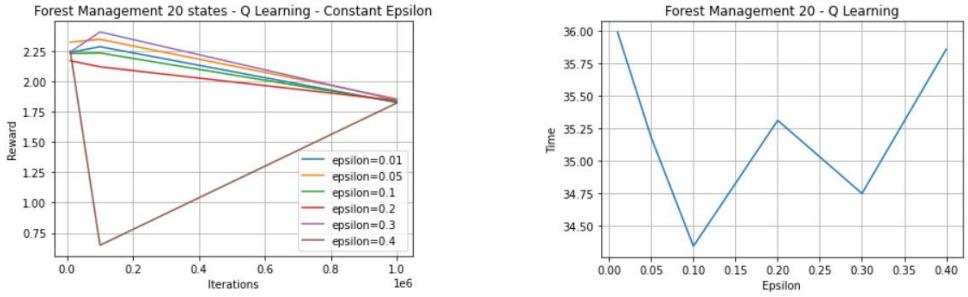


Figure 9—20 states Forest Management QL (a) runtime plot (b) average rewards plot

In the larger scale problem, similar phenomenon is observed in Figure 10, and therefore we can conclude that in forest management problem, QL is not a suitable problem, since it requires more computation time while not providing good result in the experiment.

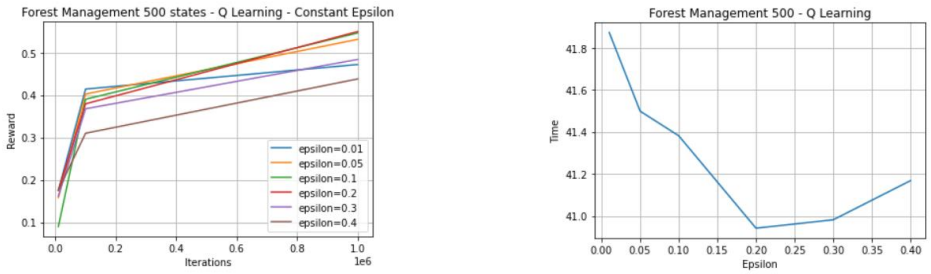


Figure 10—500 states Forest Management QL (a) runtime plot (b) average rewards plot

3 CONCLUSION

In this assignment, PI, VI and QL are implemented on Frozen Lake and Forest Management problem. PI and VI performs relatively close to each other, but PI converges in less iterations and tends to finish faster in practice, and therefore it is recommended to be implement first and have a rough evaluation of the problem. If given enough iterations and computation time, VI should converge to the similar optimal policy as PI did. On both problem, PI and VI performs better when the discount factor is larger, and that means that the agent should consider the future rewards rather than just greedily looking for maximal current rewards. QL can potentially perform better than PI and VI, but it requires more computation time and resources, and since the exploration contains the randomness, it is not guaranteed that the performance will be consistent between separate experiments. However, due to the model free nature of QL, the implementation is relatively easy and versatile to different kinds of the problem. It can be used as an alternative if given enough computation time and resources.

4 REFERENCES

1. https://gym.openai.com/envs/FrozenLake-v0/
2. https://pymdptoolbox.readthedocs.io/en/latest/api/example.html
3. https://towardsdatascience.com/policy-iteration-in-rl-an-illustration-6d58bdcb87a7
4. https://towardsdatascience.com/policy-and-value-iteration-78501afb41d2
5. http://www.incompleteideas.net/book/RLbook2020.pdf (Sutton and Barto RL textbook)
6. https://www.baeldung.com/cs/epsilon-greedy-q-learning