Markov Decision Processes

Yi Zhao

Yzhao644@gatech.edu

1. Introduction

This paper discusses methods to solve two Markov Decision Processes problems. Three methods, Value Iteration, Policy Iteration and Q-learning method were introduced to solve two MDP problems, a non-grid Forest Management Problem with larger states and a grid problem, Frozen Lakes with relative smaller number of states.

Further discussed are convergence analysis, hyper parameter's individual effect on how the Q-learning can converge to optimal policy.

2. Problems

Since we are required to use Value Iteration and Policy Iteration to solve the problems, the problem needs to be model based and a transition matrix is required for the two algorithms. It easier to form the transition matrix for distinct state problems. Model based and distinct states are two reasons for both of the below problems.

2.1 Frozen Lake (small state problem)

For the grid problem, I used Open AI Gym's frozen lake with some modification on the rewards. The states are locations on the frozen lake. Agents could take 4 actions, i.e. up, left, down, right. The grid is a square frozen lake that has several holes that agent can fall into. There is one grid which is the ultimate reward location. If the agent can reach that location, a final large reward is granted. When agent takes an action, there's only 1/3 chance that the action will be taken, since the frozen lake is slippery, the agent might end up in the other directions. For up move, agent could move to left or right with 1/3 chance.

Since I will use mdptoolbox to solve the problem with Value iteration and policy iteration, I'll need to make the reward and transition matrix transfer from gym environment to mdptoolbox. Detail could be found in section 3.1.

There are three reasons I liked about this problem. Firstly, it is a small problem with 16 states. Secondly, it is a real-life problem that is easy to interpret and relatable. Also, it is easy to visualize the result. Lastly, the agent could have multiple actions and the action is dynamic, with a probability to move to unintended directions, making this small state problem complex to solve.

2.2 Forest Management (large state problem)

Forest management is a non-grid problem working on deciding whether to cut a forest to make money or reserve it for wild animal. There is a possibility of p that the forest could be burnt down to back to initial state. States are defined by the life of the tree. The actions are cutting or reserve the forest.

Immediate rewards are 1 for cutting before final state, and 200 for final state. And 0 for reserving the forest before the final state and 400 for final state.

The reward is gain chosen by how the q learning will work with mdptoolbox.

The reasons I liked about the problem are it is a non-grid problem. The actions space is small to handle large state spaces.

Larger problem needs larger gamma for method to work. Otherwise, it's all cutting and done.

3. Value Iteration and Policy Iteration

In MDP problems, the goal is to find the optimal policy to maximize the total rewards. It focus on the total rewards instead of immediate rewards. The bellman equation is used for reinforcement learning problems.

$$V^{\pi}(s) = R(s, \pi(s)) + \gamma \sum_{s'} P(s'|s, \pi(s)) V^{\pi}(s')$$

Dynamic programming is a method to solve the problem. For both value iteration and policy iteration, the overall idea is similar. First step is to get an update of the value table and then exact policy according to the table.

$$V(s) := \sum_{s'} P_{\pi(s)}(s, s') \left(R_{\pi(s)}(s, s') + \gamma V(s') \right)$$

$$\pi(s) := \operatorname{argmax}_a \left\{ \sum_{s'} P(s' \mid s, a) \left(R(s' \mid s, a) + \gamma V(s') \right) \right\}$$

Value iteration and policy iteration are two algorithms to solve MDP problems. For value iteration, we begin with random value function. Then we step through each state and get the Q table and update the value function with the max value of Q(s,a). this continues until the improvement is less than the threshold or maximum iteration is reached.

For Policy iteration, we start with a random policy. then we use the new policy to update value table and see if the value table converges as above. If not, we update the value table and recreate a policy with the largest state action combination as above the repeat, until converges.

According to the above definition, one of the differences is that policy iteration might need more time to converge, as in each iteration, PI must finish all the policy before checking the convergence. But in terms of number of iterations needed for convergence, it is usually hard for policy to use many times to converge. Details will be shown in part below.

Another important thing to mention is how to define convergence in both VI and PI. In this problem I used two metrics for that, the error per iteration and the mean V vs iteration.

The mean V is the average for the value table, which is the state's maximum future rewards. This is a good representation of how much rewards the agent could be collect with different policy. The larger

the mean value, the better the policy. The problem is solved when there is only small change in the value table. The error is different changed per iteration. It is also a good representation on when the algorithm converges. I showed the delta convergence plot in below to confirm convergence.

For both QI and PI, one hyper parameter is the discount rate Gamma. Plots are also created to show as the gamma increase, the iteration to converge increases.

3.1 Frozen lake problem

For frozen lake problem, I transformed open AI Gym's environment to the P and R table for the mdptoolbox. The changes include immediate reward and P and R tables.

The immediate reward is the expectation of the actions that the agent could take. For example, if the move makes the agent reach to the ultimate reward, which could award agent 10 points, the reward for that action is given by 1/3(10+0+0).

My modification from the gym environment is change the ultimate reward from 1 to 10, this is to encourage agent to go further to reach the ultimate state. Also, there's a -5 penalty for agent to fall in the hole and -0.5 for staying on the path. This is set up for agent to reach the ultimate location as soon as possible. Also for mdptool box's Q-learning algorithm implementation, there's no terminal states but a state reset for every 100 steps. It is important to have the agent reach to goal as soon as possible instead of staying on the path and not moving forward.

For the transformation part, I create the P table with dimension as action* state * state(4*16*16), with probably as 33% for each action. For R table, the shape is state*action, with each reward as the expectation of the state with the action.

Below is the detail result for solving a 4 by 4 frozen lake grid with rewards as (10,-5,-0.5) and transition matrix as 1/3. Some key findings are as follows.

3.1.1 Frozen lake (small problem) VI and PI solution analysis

Both value iteration and policy iteration converge and they converge to the same optimal policy. The numbers in the tuple represents left, down, right, and up actions.

Frozen lake optimal policy and visualization.

```
(0, 3, 3, 3, 0, 0, 0, 0, 3, 1, 0, 0, 0, 2, 1, 0)
```

S left	F up	F up	F up
F left	H left	F left	H left
F left	F down	F left	H left
H left	F right	F down	G left

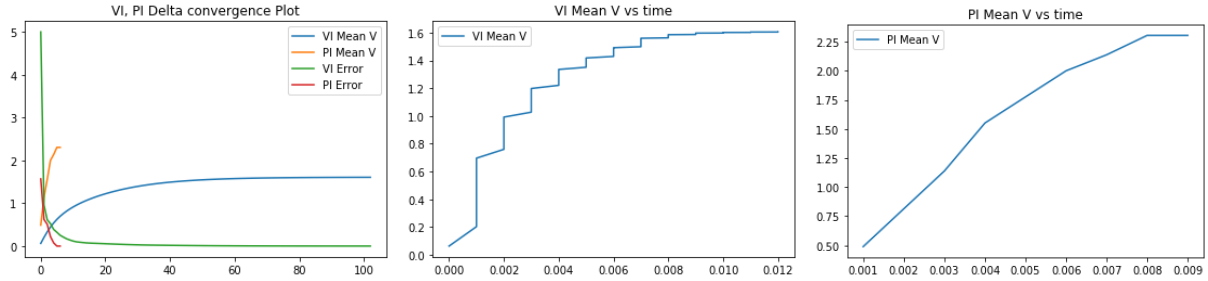
The above policy makes sense to me. In the beginning, the agent are moving left to avoid the two holes in the right part of the maps. It chooses the left action instead of down movement to avoid a 33% accidental slippery move to the right and end up to the right side of the map. If the agent accidentally

move the right side of the first row, it will try to go back to the left side of the map by moving up and crashing to the wall to avoid any chance to accidentally move down to fell in the second row's hole.

Another interesting point is state 6, which is surrounded by two holes. In real life, one could probably decide to move down and reach for the ultimate goal. But the result is move left. This makes sense since move either up and down has a lower expected reward as there are double chances for the agent to fall to either side and get a penalty.

For the state 14, the agent chooses to crash down to the wall again to avoid accidentally moving up to fall into the hole at the state 11.

3.1.2 Frozen lake (small problem) VI and PI comparison



The first plot is the delta convergence plot to show both PI and VI are converging in the end. You can see the error and mean V both flatten out at the end.

The difference between VI and PI are that VI needs more iterations to converge. PI needs only need 7 iteration to converge while VI takes 104 times.

PI needs less time for calculation than VI as well. But the time need for PI to converge is three quarter vs VI time, 0.009 seconds vs 0.012 seconds, while VI has 15 times number of iterations more than PI. This is due to the problem being a small states problem. Still PI needs more time per iteration.

3.2 Forrest Management problem

For forest management problem, I use state as 400 to make it a large problem. The fire probably is 0.1, which means every time we reserve the forest, there is 10% chance the forest will be burnt down. The final reward for reserving to the final state is 400 and 200. This is chosen with the mdptoolbox setup. As there is not terminate state and gamma is set up to be 0.999. after 400 discounts, the reward is still worth agent to go for.

I have also updated q learning source code to generate random state for every 400 steps.

3.2.1 Forest Problem (large problem) VI and PI comparison

Both VI and PI solve the problem by converging in the end. They both converge to same optimal policy where there are 336 cut actions and 64 reserve actions. The policy is reserve for the start state and cut for the first 337 state and reserve for the last 64 state.

(0,1,1,1,…..0,0,0)

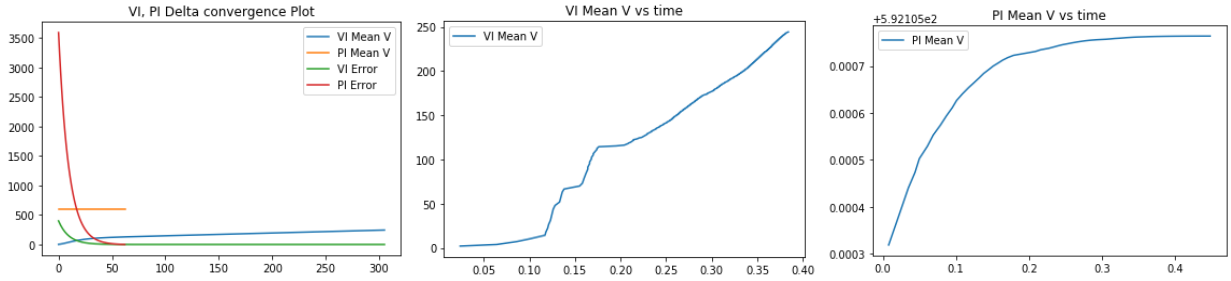
This result makes sense to me. Since agent would collect all the first small reward by cutting, unless it reaches to the last final states, where the agent has a chance to reach the final state without getting the

forest burnt to starting state. This is the result of making the rewards as 400. If we increase the reward to 40000, the optimal policy stays the similar way but has more reserve actions towards the end.

3.2.2 Forest management (large problem) VI and PI comparison

Here I plot the delta convergence plot again to prove the convergence. Similar result as small states problem is observed where both of the mean V and error metric are flattened out in the end. But compare with the smaller state problem, VI and PI both needs more number of iterations to converge. This is due to the large action space that require the algorithm to calculate more times.

For time comparison, here we can see the difference between PI and VI. This time PI needs 10% more time to converge than the VI and VI has 6 time more number of iterations to converge. This is again due to the large problem having more spaces.

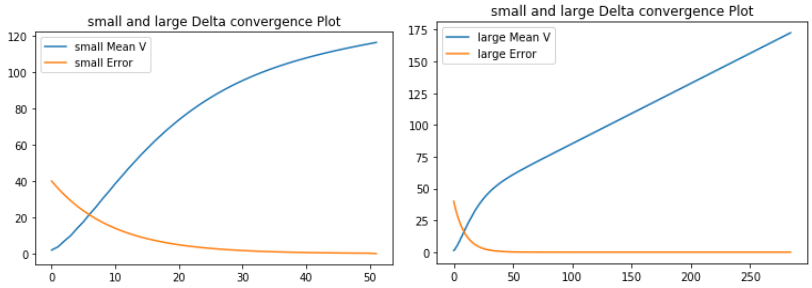


3.2.3 Larger forest vs smaller forest

Here I created a smaller forest to compare error and mean value to see the convergence.

Small problem takes about 0.05s and 50 iternations to finish while large problem needs 0.28s and 270 iterations.

Delta convergence plot is as below.



3.3 Hyper Parameter for VI and PI

As mentioned earlier the discount factor, gamma, is a hyper parameter that can affect the solution of the problem. Another parameter I take as a hyper parameter is the ultimate reward that can affect the training problem.

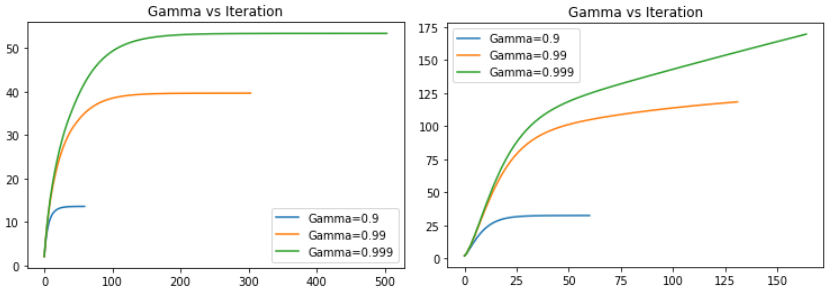
3.3.1 Gamma has an effect of the problem.

Below plot is value iteration result for mean v vs number of iterations. I used three gammas to check the effect on the learning problem.

The first observation is all the problems converge at different number of iterations and total rewards.

With a higher gamma, the agent would have a higher horizon to the future, make the problem more complex to solve. As a result, higher gamma the green line all converge later.

Also the total reward is different, as agent could pick up more rewards in the future.



3.3.2 Rewards can also play a part in the convergence problem

This has also been discussed earlier. Frozen lake's ultimate rewards has been increase to 10 and forest management is also increased to 400 to improve algorithm's performance.

	rewards	Avg convergence iteration	Avg Convergence time
frozen lake	1	178	3.5
	10	120	5
forest management	4	30213	22
	400	28729	30

4. Q-learning Algorithm

I chose Q learning as my favorite algorithm to solve the problem. Q learning has some nice features to be my favorite. It is a off policy and model free algorithm, which means you don't need transition matrix for the learning part and this is very convenient for stochastic problems like Frozen and Forest Management. It is also a greedy method, which is easy to explain when agent need to take an action and easy to relatable to the real-life problem. Below is the bellman equation to represent the algorithm.

Q-learning requires a q table for learning process, with large states and specially large actions combination, it could hard to create and update the table. This is not a problem for our case, as we only have 2 and 4 actions.

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

$$\underbrace{\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad}_{\text{new value (temporal difference target)}}$$

With the greedy method, there are many moving pieces that needs to be tuned for the problem. Like learning rate alpha. Larger alpha means a faster learning pace, but it could cause the problem to miss global optimum and stuck with local. The alpha is decayed linearly to a minimum alpha, by a decay factor. This is to help the convergence.

Also there are different exploit and explore strategies for q learning to choose. Exploit meaning the q learning to be greedy and take the action to gain the max future rewards. Explore is to take a random action to update q table. This is controlled by a hyper parameter epsilon. Whenever an action is needed, a random number from 0 to 1 is generated, if the number is larger than epsilon, we exploit, otherwise, explore.

Some strategy examples include greedy method, epsilon greedy method, and a combination of both. With different strategies, there is another parameter epsilon to tune. Also the decay rate of the epsilon. Epsilon begins as a large number so that agent could have opportunity to explore the actions at different states to try to reach to the ultimate goal. It then decay to smaller and focus more on exploit to get the largest future rewards.

The discount rate also needs to be tuned in this problem to get the action focus on the right length of the history to get the algorithm to converge.

Without proper hyper parameter tuning , Q learning won't converge or will converge to local optimal

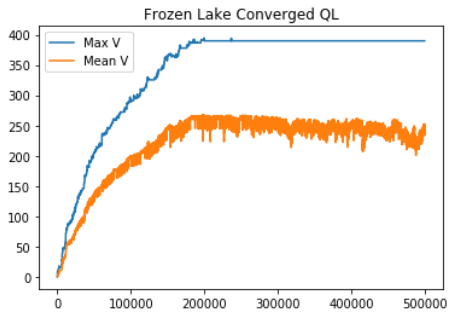
4.1 Frozen Lake Q learning

4.1.1 Frozen Lake result

With the proper tuned hyper parameters (process will be discussed in next section). The algorithm converges at the end.

```
(0, 3, 3, 3, 0, 0, 0, 0, 3, 1, 0, 0, 0, 2, 1, 0)
```

S left	F up	F up	F up
F left	H left	F left	H left
F left	F down	F left	H left
H left	F right	F down	G left



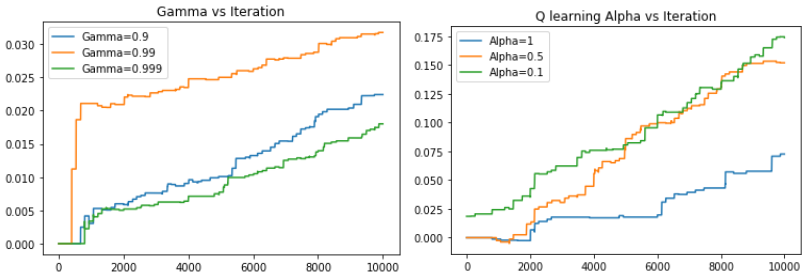
The time and iteration numbers needed has a huge difference. Q learning needs a lot of exploration to reach the optimal policy. This is a trade off by not having the transition matrix. But in the model free settings, q learning would have a great advantage.

	Policy Iteration	**Value Iteration**	**Q learning**
Time To Converge	0.09	0.012	32.5
Iteration Needed	7	102	172500

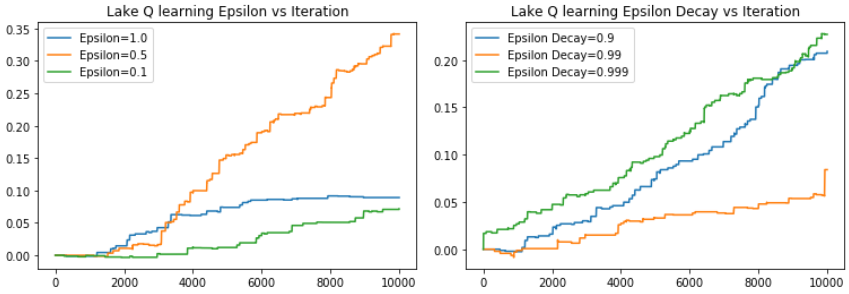
4.1.2 Frozen Lake parameter tuning

Below is the hyper parameter tuning process. The parameters to discuss are discount rate gamma, learning rate alpha, exploration strategy decider epsilon and the decay rate for the epsilon to change.

The q learning are run for only 50000 iterations to show the performance.



From the plot, a gamma of 0.99 has be best performance. As mentioned in the earlier section, I've updated the MDP's q learning code to restart when agent reach to the final station. Given the state space, it's not necessary for agent to maintain a longer memory, so a middle point 0.99 is chosen.
For learning rate, 1 has a really lower mean value. This proves my assumption that a large learning rate could cause the algorithm to miss the optimal policy. There is not much difference between 0.1 and 0.5. I've choose a larger 0.5 for as faster learning.



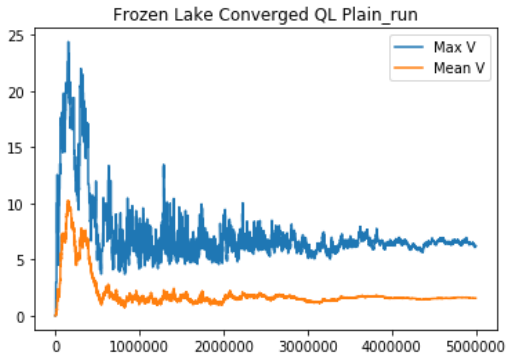
Above two exploration strategy hyper-parameters training. It turns out with combination of epsilon as 0.5 and decay rate as 0.999 has the best result.
A medium epsilon means the algorithm does not need a large exploration to reach a good result, this could because there's not many states to explore to reach the ultimate goal. With a higher decay rate, the algorithm tries to slow down the speed exploration shrink, to exploit more after exploring for a while.

4.1.3 Frozen Lake Exploration Strategy analysis.

I have tried a combination of hyper parameters to come up with the final optimal policy. Three exploration strategy are implemented during the experimentation. They are greedy method, epsilon-greedy and a combination of both.
The best strategy is the combination of both. This is inspired by the epsilon and epsilon decay hyper-parameter tuning. You want the agent to explore for new rewards, but you don't want them to go explore again immediately. So my final strategy is to explore every 1000 iteration and during the iteration exploit the max state action combination.
Below is one of the failing strategy. The algorithm manage to converge in the end, but it has a lower mean V (almost 0) and converge to a local optimal.
To visualize the result, the agent ends up turning up and afraid to got down. This could due to the epsilon converge too fast and agent stuck at the first rows and refused to go down.

```
(3, 3, 3, 3, 0, 0, 0, 0, 3, 1, 0, 0, 0, 2, 1, 0)
```

S up	F up	F up	F up
F left	H left	F left	H left
F left	F down	F left	H left
H left	F right	F down	G left



4.2 Forrest management Q learning

4.2.1 Forest Management result

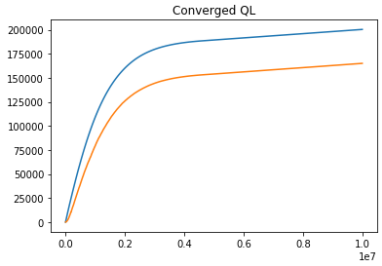
With the proper hyper-parameter combination, the q learning algorithm converge. But it doesn't converge to the VI and PI result. But it is very close to them, about 80% of the strategy are same with VI and PI.

The optimal q learning has about the same action of cutting the tree, 320 cutting action and 80 reserve vs 336 cutting and 64 reserving for PI and VI.

My understanding of this is q learning has not reached to the end process very often to experience the ultimate result, so it takes random actions for the last few steps to cut the wood. May be a larger reward could help to avoid this problem. This is a good area for future study.

Below is the delta convergence plot for forest management problem and time and iteration comparison between vi and pi.

Q learning really need a lot of time and iterations to achieve a convergence and close to optimal solution. One thing to mention here due to the large gamma, the mean v value here is not comparable between vi and qi as q learning is keeping all the rewards in the history, larger number of iteration results in larger mean value. But the overall converging trend is still valid for q learning.



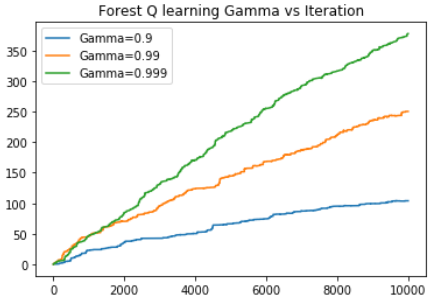
	Policy Iteration	Value Iteration	Q learning
Time To Converge	0.45	0.4	160
Iteration Needed	52	305	3000000

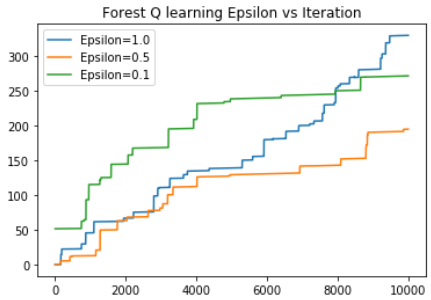
4.2.2 Forest management Hyperparameter Tuning

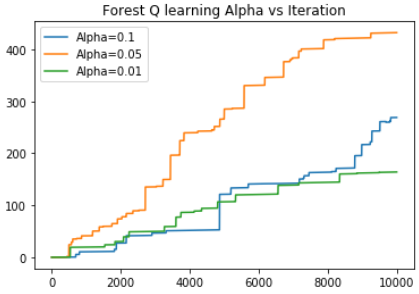
Same approach for hyper-parameter selection process is implemented for forest management.

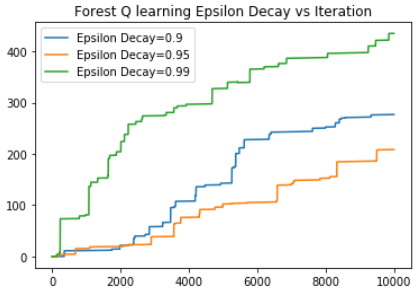
The difference between large problem and small problem in hyper-parameter selection, I chose a larger epsilon and smaller decay rate for the q learning to explore more.

The alpha is also smaller for a slower learning process to catch the global optimum.



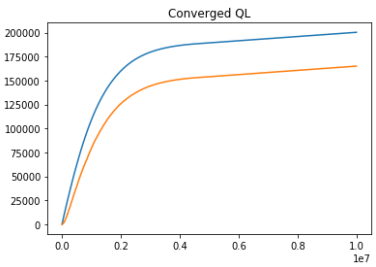


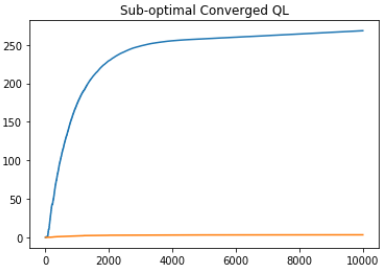




4.2.3 Q-learning Exploration strategy

I used three exploration strategy. Greedy, epsilon-greedy and combination of the both. Compare with the optimal policy on the left, right policy converges but doesn't generate larger mean value due to an earlier converge to miss the global optimum and making the agent to stuck at earlier rewards.





5. Summary

For this assignment, I created two MDP problems, Frozen Lake and Forest Management. I used PI, VI and Q-learning to solve the problem.

Both problems are solved and converge to same optimal result for VI and PI. Q-learning also converge to global optimum policy for Frozen lake while Forest Management has a really close to optimal policy.

VI converge faster than PI in terms of time, but needs more iterations to converge, while Q learning requires more time and number of iterations to converge.

Hyper-parameter tuning is highly important for Q learning, otherwise it could converge to sub-optimal solution.