Markov Decision Processes

Exploration of MDP and RL algorithms

Roshan Gajurel
Georgia Institute of Technology
Rgajurel3@gatech.edu

Abstract—This paper explores two different Markov Decision Processes of varying complexity using value iteration and policy iteration and compares the results. We will also select Q-Learning as our reinforcement algorithm to solve both MDPs and compare the performance.

I. Introduction

This paper focuses on the following algorithms to solve Markov Decision Processes:

- Value Iteration
- Policy Iteration
- Q-Learning

We will explore these algorithms using two interesting MDPs and analyze and compare them. Pythons gym library is used exclusively to create environments. Before we begin the comparison let us understand what MDP is, and how these algorithms work.

II. Markov Decision Processes and Algorithms

A. Markov Decision Processes

Markov Decision Processes is a framework for decision making in an environment that is random or stochastic. In a stochastic environment an action taken might not result in the desired outcome with some probability. The environment in an MDP is defined with the following properties.

States(s): Set of position in an environment which does not change.

Action(a): Set of action that can be taken in a state

Transition Function (s, a, s'): It is the probability you end up in **s'** given you are in state **s** and take action **a.**

It only depends on the current state **s**.

Reward R(s): Reward or value for entering in a state

Policy: Is a function that returns an action given a state **s**

Optimal Policy: It is the Maximum reward that you can get by taking actions in states in the environment.

Utility U(s): Utility of a state when following policy p is the discounted sum of all the rewards that will be accrued when the policy is followed. It can also be defined as a long term or delayed reward.

Discount Factor: It decreases the value rewards for future states.

MDP is based on the idea of Markovian property states that in stochastic or random process the conditional probability of a future state depends only on the present state. Also, it is important to note that, in an MDP the environment is stationary which means the state and rewards are constant and do not change. Also, we always know which state we are on and the reward for the state.

B. Value Iteration

Value Iteration is algorithm that calculates the optimal policy by calculating the utility for each state in such a way that the policy can be calculated by taking the action that takes the maximum value for each state. Value iteration does this by first assigning random value to each state it then updates these values based on all the states it can reach or their neighbor. For example, a state close to the goal state will have a higher value and it will increase on iteration. Since the value of states is depended on the neighboring states the higher value propagates to the neighboring states creating a path towards the goal. This is repeated until it cannot find the values for which the utility does not get better. This is defined by a certain threshold. The utility of each state is calculated by using the below Bellman's Equation:

$$U(S)_{t+1} = R(S) + \gamma \max_a \sum_{s'} T(s, a, s')\ Ut(s')$$

Value equation can be slower because it iterates at each state to calculate the utility using the discounted reward. It is important to note the difference between the utility at a state and the reward. The reward is simply the gain at that certain state where as utility takes into account the reward of all future states given it follows the policy which is discounted by using the discount factor.

C. Policy Iteration

Policy Iteration is similar to value iteration because it uses the Bellman's Equation to find the optimal policy, but it does it in a different way. Instead of assigning values to each state and iterating to find the best utility like value iteration it rather starts with a policy and computes the utility for the policy until the optimal policy is found. The algorithm for policy iteration works as follows:

- First it generates a random policy from the set of available policies
- It then calculates the utility of the given policy
- It improves on the policy by
    - updating the policy when it finds a better action for a state
    - repeat until the best policy is found

This is more expensive then value iteration.

D. Q-Learning

Q-Leaning is a family of reinforcement learning algorithm. Like policy and value iteration it also tries to find the optimal policy by maximizing the reward. It does this by using transitions to come up with the Q equations. It is more of the learner than policy iteration and value iteration since it requires far less

information about the reward and transitions and is also called model free. It only requires all possible states and the actions that can be taken. Q-Learning starts by choosing a random value called Q value for each state. Based on the reward it receives for visiting a state the Q value is updated. At each state the learner might take an action for which it does not know the reward to, this is called exploration. This might be an issue if it keeps on exploring and never exploiting what it learns. Q-learner strikes a balance by exploring initially and learning in the later phases.

III. MDP Problems (Frozen Lake)

I selected the frozen lake problem for the analysis of these algorithms for its simplicity and generality. Let's describe the problem in a real-world situation first. Imagine there is a large frozen lake and you need to get to the other side of the lake. This frozen lake, classic MDP problem, is mapped in a grid. The grid has a starting state which is where you will enter the lake, an end state which is where you are trying to get to. The other states in the grid are either frozen or has a hole. Needless to say, if you step on the hole you die but like a game you get to try again and hopefully have learned not to do that again.

At each position in the grid you can take four action, you can either go up, down, right or left. We will make the transitions stochastic when analyzing them using value and policy iteration. In terms of our problem, the ice is slippery so if you take an action in a state there is only 1/3 chance that you will get to the desired state and 1/3 that you will get to either of the right angles where you intended to go. This makes the environment random. It is helpful because it forces the us to explore and take new paths, although accidentally, rather than staying on the same path for which we know the rewards for. While exploring new paths, the algorithm might discover states with higher rewards that it might not have found otherwise if the environment was not random.

I find this problem interesting because it is at the core of what MDP and RL is trying to solve and at the same time it is simple enough to understand. This problem can be extended to solve real world problems like path finding in robotics or kidnapped robot problem. These algorithms enable us to program these robots in such a way that they learn over time instead of having to code all the paths and obstacles for each problem.

For the purpose of this analysis we will use a smaller frozen with a grid size of 4x4, which is a simpler problem. We will then move to a larger frozen lake with a size of 15x15.

IV. Algorithms Comparison

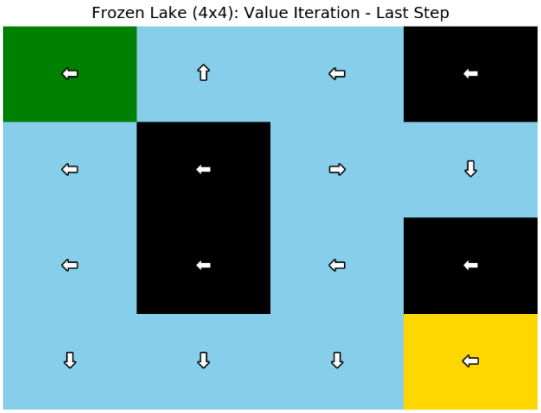
There are certain parameters that we influence how the algorithm behaves. Let us discuss those parameters. Step Reward is the reward for being in the frozen state. This is set to **-0.1**. Hole Reward is the reward for being in a hole. This is set to **-1**. Goal Reward is the reward for getting to the goal state which is set to **0**. Arriving at either the goal or hole will end the iteration with a value of 0 or -1 respectively. Also, the transition probability of ending up in the desired state is set to 1/3, later it will make comparison of this with other values. Changing this will make the decision we take more or less decisive at any given state. In other words, this decides the exploration versus exploitation decision in the MDP.

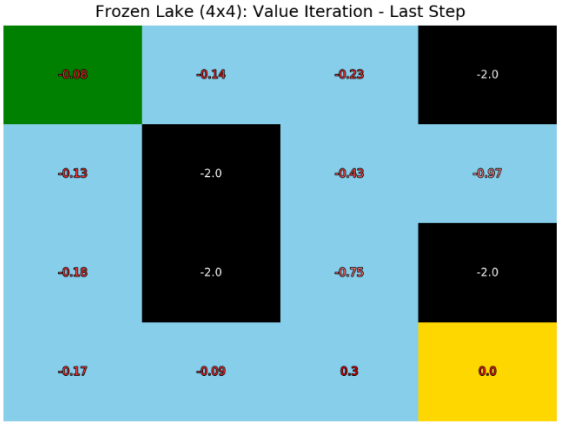
A. Small Map

Let us first compare these algorithms in a small map.

Value Iteration

For value iteration I wanted to compare the outcome when the probabilities to get to the next state are changed. Below is the diagram of us look at optimal policy for the smaller map with 1/3 probability of getting to desired state.

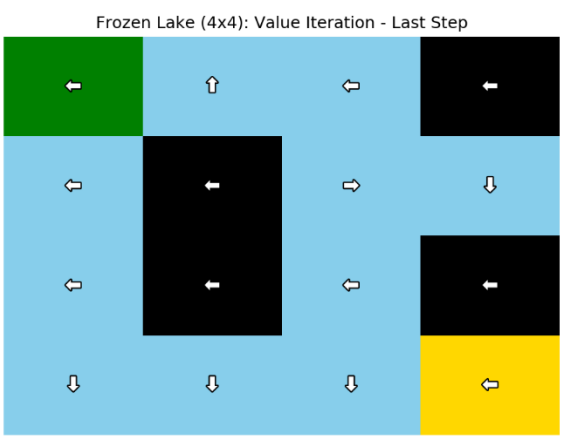


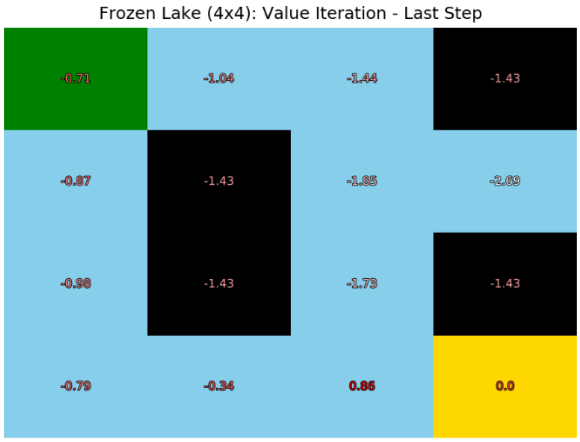


Let us first understand the diagram. The green grid is the start position, gold is the goal position, black grids are the holes and blue grids are the frozen. So, the objective is to get from the green grid to the gold grid without falling on the black grid. The arrows on the diagram on the left represents the action that should be taken at each state based on the optimal policy. The diagram on the right represent the utility values for going to the next state from a state.

One thing we notice immediately is that the arrows are facing the outside of the grid at instead of the direction it should actually go to. This is because of the non-deterministic nature of the world. If the world was deterministic the arrows would neatly start from the green and end up in the gold. However, since there is a 1/3 probability that it might go to any of the right angles, it is safest if it avoids the black holes. If we look at the values in the grid on the right, we can actually see why it is making those actions should be taken at each step. Suppose we are at the 2x1 position of the grid which has a value of -.13 we can see that since all the values around it are negative it makes most sense to go left there in which case there is 1/3 probability to go up or down which has the combined utility of -0.08+(-0.18) which is the most reward it can get at that point. If it turns to any other direction, there is a chance for it to get to the black box which has a utility value of -2.0 and the combined reward would be greater than turning left.

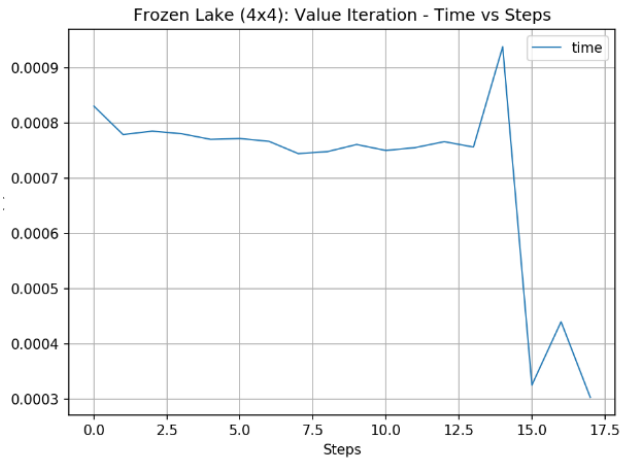
Just for this small value map I adjusted the probability of getting to the next state from 1/3 to 0.9 because I thought this comparison might be interesting.

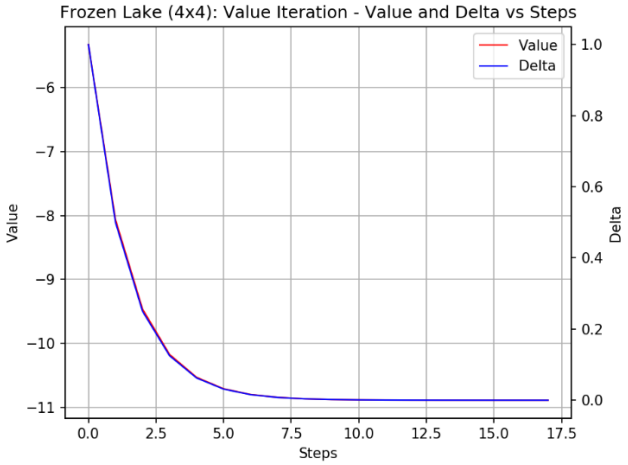




We can see that there were no changes to the policy however the utility values are different for each state. Looking at this we see that it does not matter what probability we choose but once we look at the performance we can see a clear winner.

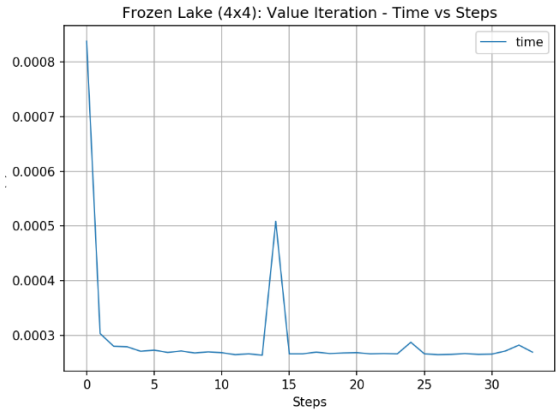
Performance:

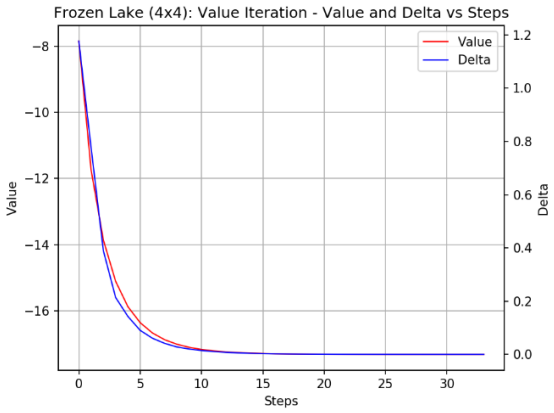




We can see that the it converged in 18 iterations (since the graph starts with 0), and the maximum time it took was on the 15th iteration taking **0.0009s** and the total time taken was **0.012778s** for convergence. On the right we see the graph of value and delta against steps. Here value is the reward at each iteration. Since we set negative rewards, we can see that the rewards decrease. Delta is the change in the utility at each step compared the previous step. The algorithm converges when this delta is smaller than a threshold.

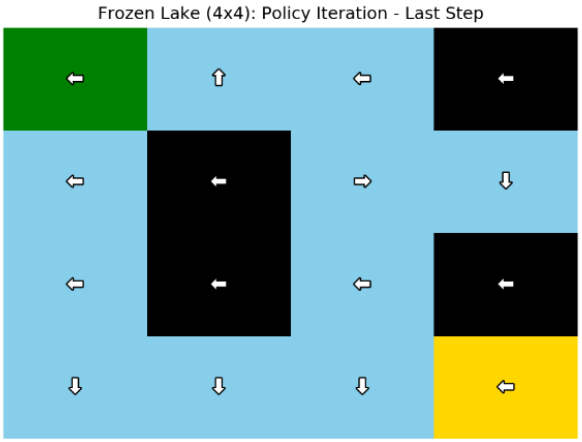
In the below graphs I changed the transition probability to .9 from 1/3 and we can see that it actually increased the number of iterations for convergence for the small world.

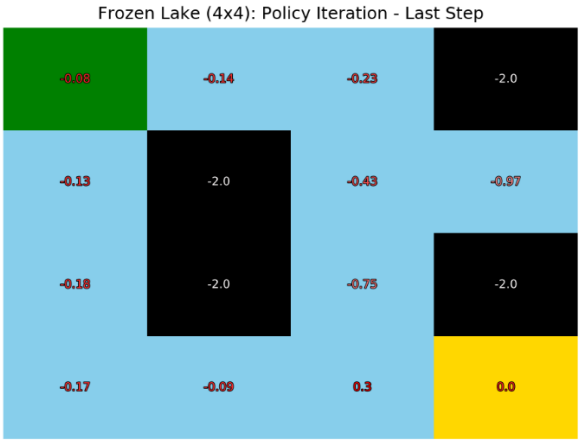




Policy Iteration

The below diagram represents the outcome of policy iteration. We can see that the policy is exactly the same as with value iteration, however it has different utilities than value iteration at each state. Which makes sense is because policy iteration calculates the utilities differently.



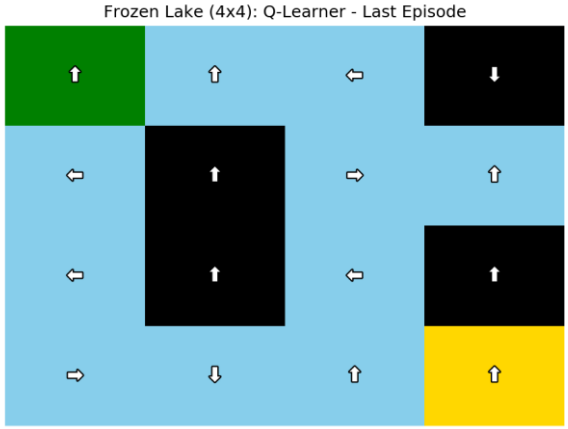


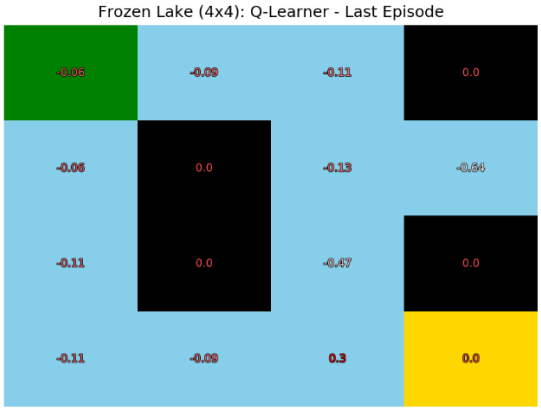
Performance: For the easy map, since the policy converged in just **two** iteration the graph for this is not very intersting. However it is interesting to note that it took policy iteration **0.015385s** to converge in **2** steps. Which is higher than value iteration. So, we can observe that even though policy iteration converges in fewer steps it takes loner to run each iteration. This is because policy iteration first guesses a path and then adjusts the values by iterating on each policy, so it learns more at each step, but it takes longer.

Q-Learning

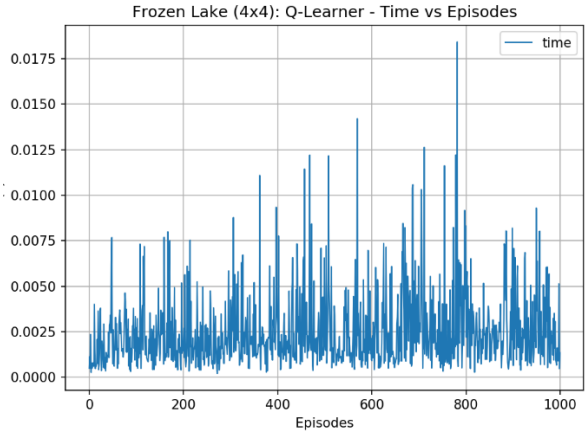
Q learning is different than value and policy iteration because it does not require a model to learn. It is a benefit because it mimics the real-world situation more clearly. For the purpose of this analysis we are going to use the epsilon greedy approach. We used alpha, epsilon, epsilon decay and discount factor as hyperparameter for this analysis. To find out the best hyperparameter we changed all these values and found the optimal solution based on maximum reward. However, when we observe the learner in this

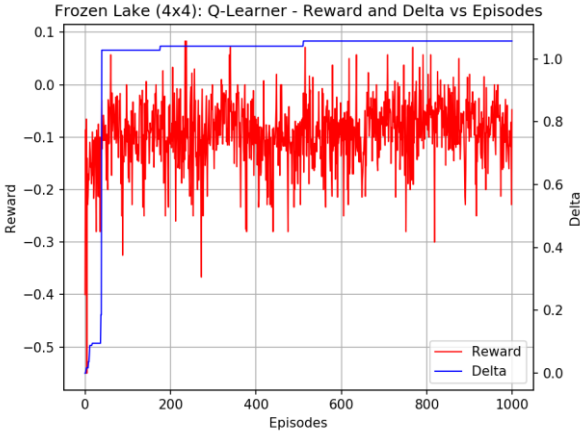
scenario, we see that it does not perform as well as the other learners. This is mainly because Q-leaner does not use the domain knowledge like value or policy iteration algorithms.



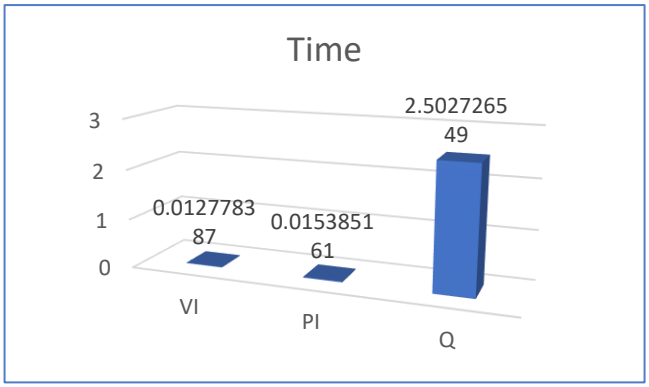


We can see that it got most of the policy right but not as accurate as PI and VI and it does not converge either. This is because it does not have as much domain information as other learners. It also suffers from exploration-exploitation dilemma.





Q-Learner aslo takes significantly more time than other learners because it needs to explore more to come up with the otimal policy. Obeserving the reward and delta versus episode graph we see that it learns quickly initially but it takes a while to get better. We also see that the delta never gets better and the reward also fluctates and does not get better on further iteration. Since the maximum allowed iteration in this case is 1000, it does not converge and the best policy of all these steps is picked.

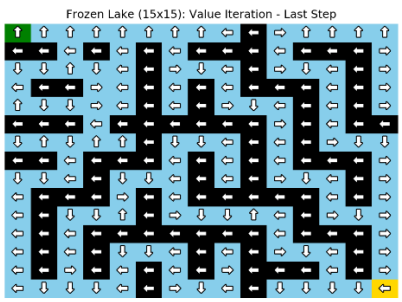


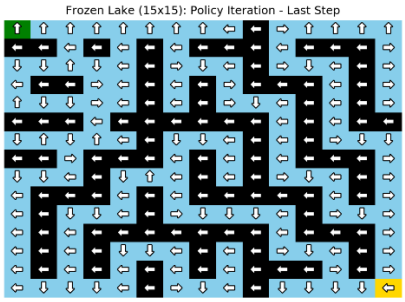
The above graph shows more detailed comparison of reward and time. We see that VI has highest reward and is also the fastest. Although Q-Learner mimics the real world more than both these algorithms, by a slight margin Value iteration is a better learner for this environment.

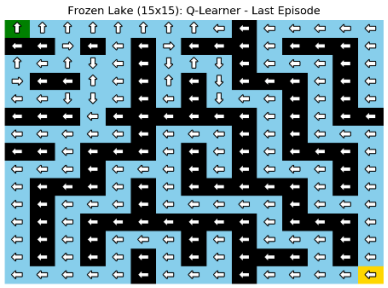
V. Comparison of algorithms in a Large frozen lake of size 15x15

Since we have interpreted the graphs for the 4x4 frozen lake problem and we can better understand the graphs and the outcomes, let us compare just compare all three algorithms side by side for the larger map.

First lets us compare them based on policy.

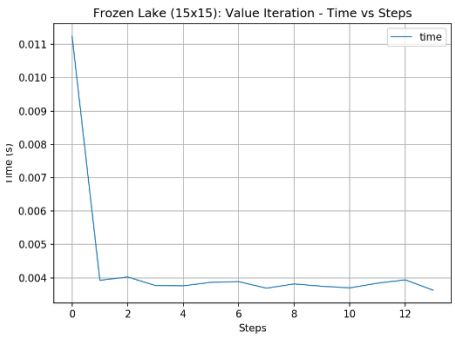


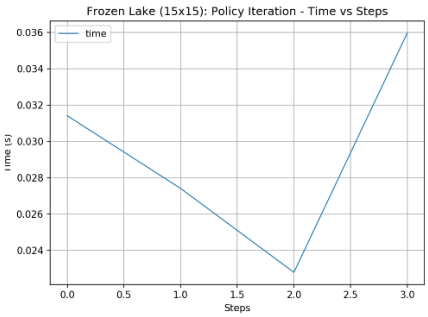


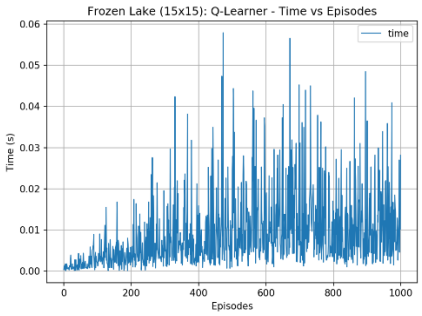


Like the smaller frozen lake problem, for larger frozen lake, value iteration and policy iteration have almost the exact same policies. If we look carefully, we can see some differences. This is interesting because they use two different strategies to find the optimal policy, however it makes sense because both the graphs converged and, in this case, they happened to converge almost the same policy. Q-Learner on the other hand is completely lost again. It did do well initially to compute the utility values of the states near the start, but it eventually reached the maximum allowed iteration of 1000 and it was not able to converge. It probably would have eventually converged if it was allowed unlimited iteration.

Now let us compare the steps with time for all algorithms.

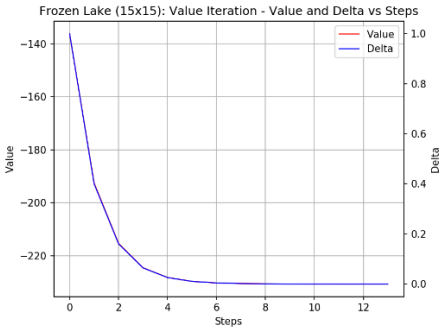


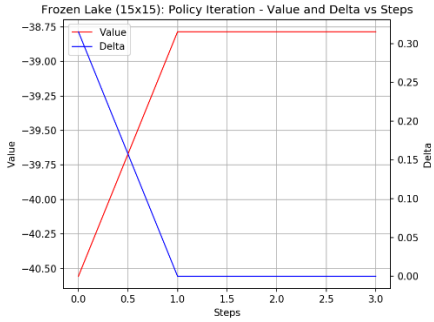


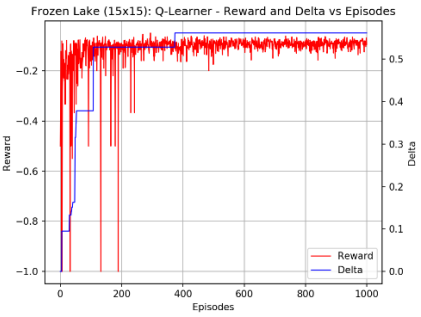


This is interesting as well as we can see that value iteration converges in 14 steps, and policy iteration converges in the 4 steps. Also, this a much more difficult problem than before and PI finds a different utility every time. But since it has almost the same policies as value iteration it did do well. However, Q-Learner like we discussed before did not converge and did not find the best policy either. In terms of time again, Value Iteration performs the best.

Now let us compare the rewards and delta for these algorithms.

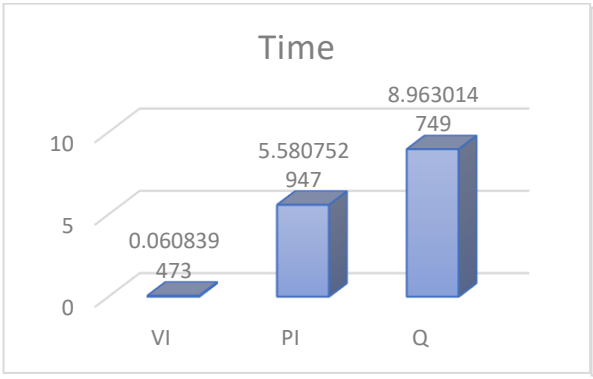


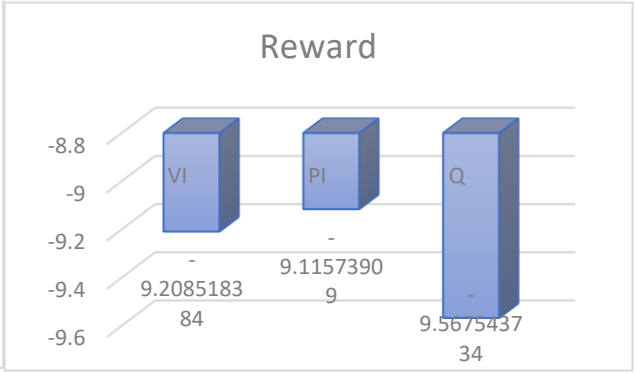




The graph for value is very similar to before, the value is close to delta and it eventually converges. For policy iteration we see that the value and delta does not change at all after the first few steps at all. This is because delta gets to 0 and it cannot improve at all after that. In the case of Q-Learner, we see that the rewards fluctuate a lot but, in the beginning, because it is learning or exploring in the beginning but then eventually it starts to exploit what it has learned. This can be seen clearly in the graph above.

Let us compare the performance of these algorithms against each other for the larger frozen lake problem.





There is so surprise in the time comparison. Value Iteration has the fastest time and policy and Q-learner take increasingly more time. In terms of reward for the optimal policy, it seems like PI has the highest reward. However, in this more complex scenario, policy iteration does better than both other algorithms in terms of reward.

Conclusion:

We can conclude that value iteration and policy iteration perform better than Q-Learner in solving these kinds of problems. This is because both value and policy iteration have a more information of the environment in terms of domain knowledge compared to Q-Learner. Q-Learner performs better in environments where there is less restriction. If it had a looser reward model for the states, Q-leaner might have converged. However, in cases like this where we have access to domain knowledge it is better to pick policy and value iteration. They perform almost the same, if speed is the factor the definitely pick Value iteration for more complex problems Policy iteration may be better.

REFERENCES

[1] OMSCS: Machine Learning- Assignment 4 https://github.com/cmaron/CS-7641-assignments/tree/master/assignment4