Markov Decision Processes

Zahid Biswas

Frozen Lake MDP problems (2 Sizes):

Frozen lake is a variation of the classic Grid world problems. The main idea is that the square shape lake is consists of certain number of states in a grid, and the problem is to avoid the holes or soft ice to reach the goal state. The agent can make four types of moves, up, down, right, or left. Probability of staying on the next state ice after an action is take is 80%. There is a 20% chance that the agent will slip to one of the neighboring states. Although it is a fictitious problem, but in real life many games such as Syntaxity, Haiku, AE puzzles, and so on uses the underlying technique to find the best path to the goal. Besides, it gives us a way to visually understand the effectiveness of each algorithm that we will implement.
For the purpose of this project we used two different sizes of the lake. Small lake has 64 states **(8 x8)** and large lake has 625 states **(25 x 25)**. The agent starts from the top left corner (marked blue in the figure 1) and moves toward the goal state at the bottom right corner (marked green). The holes (marked red) are scattered randomly to add complexity to the problem. The agent will get -1 reward if it terminates in a hole or +1 reward if it reaches the goal state. For each action the agent receives an immediate reward of -0.1.

Three Algorithms and Implementation:

1. **Value Iteration**

   Value iteration is a technique to find the optimal policy of an MDP process by finding the optimal value function. The algorithm starts from an arbitrary terminating state and work backward to find the optimal value function. It uses bellman equation as the update rule at every iteration. The complexity of each Iteration is O(n^2m).

2. **Policy Iteration**

   Policy iteration is also another way to find the optimal policy of an MDP process. Unlike value iteration, it starts with an arbitrary policy and compute the utility at each iteration to check whether the policy can be improved. If it does, the iteration continues until the optimal policy is attained. In short, it starts and store an arbitrary policy, and progressively updates the policy over each iteration to reach the optimal policy. It guarantees to find the optimal policy but with a higher complexity in each iteration, which is O(mn^2 + n^3).

3. **Q-learning**

   The goal of Q-learning is to learn a policy, which tells an agent what action to take under what circumstances. It does not require a model (hence the connotation "model-free") of the environment, and it can handle problems with stochastic transitions and rewards, without requiring adaptations.[3]

   Implementation

   We mainly used Python gym library from OpenAI[2], to implement all implement all three algorithms to solve the corresponding MDP problems. Most of the coding was done based on the work of Chad from previous semester[3].

Small Frozen Lake Problem Analysis:

For the smaller space problem, we used the following parameters for to reach the optimal policy:

	Max discount factor	Min discount factor	alpha	q_init	epsilon	epsilon_decay	Max Steps
VI	0.9	0.1					100
PI	0.9	0.1					100
Q	0.9	-	0.5	random	0.5	0.0001	1000

Table 1: Parameters used for the optimal policy for different algorithms

Although we ran over 100 experiments with different parameters values, the above mentioned are used for the optimal solution. Rest of the results are not recorded here for the space constraints. The followings are the states we found at optimal policy:

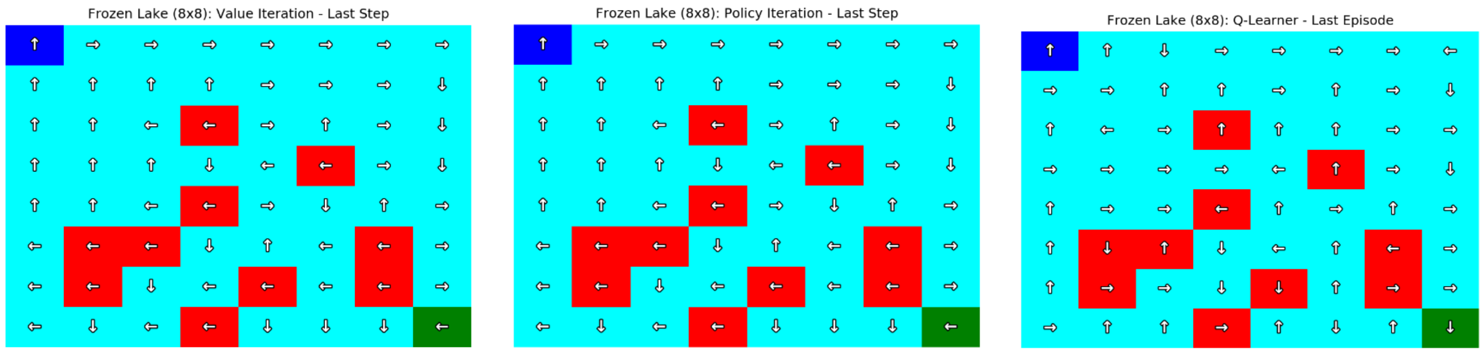

Frozen Lake (8x8): Value Iteration - Last Step
Frozen Lake (8x8): Policy Iteration - Last Step
Frozen Lake (8x8): Q-Learner - Last Episode

Figure 1: States at the last step of each algorithm after at convergence

The followings are the performance comparison of each algorithms:

Algorithms	Steps to Convergences	Time (s)	Converged	Reward
VI	52	0.04	YES	3.615

PI	3	0.104	YES	3.615
Q-learning	1000+	1.005	NO	1.001

Table 2: Performance comparison of different algorithms

In terms of optimal policy, we can see from the figure 2 that PI and VI has reached to the same last step after convergence. Whereas, Q learning even after 1000 iteration did not reach the convergence due to its exploration nature. In terms of duration, VI is the fastest to reach the convergence, although PI takes much less steps than all other algorithms.

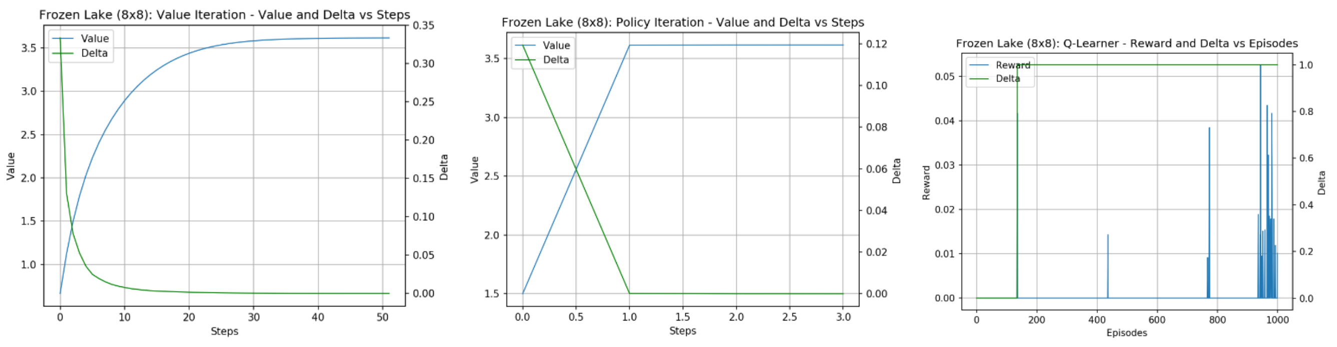


Figure 2: Reward delta vs steps graph for different algorithm for small lake problem

From the figure 2 we can see that after convergence both PI and VI is still increasing in reward at very small rate. Change of reward value is almost 0. For Q learning we can see even after 1000 iterations the delta is still fluctuate at a higher rate. With increasing episodes, the fluctuation of this delta is increasing. In other word more exploration possibility is visible.

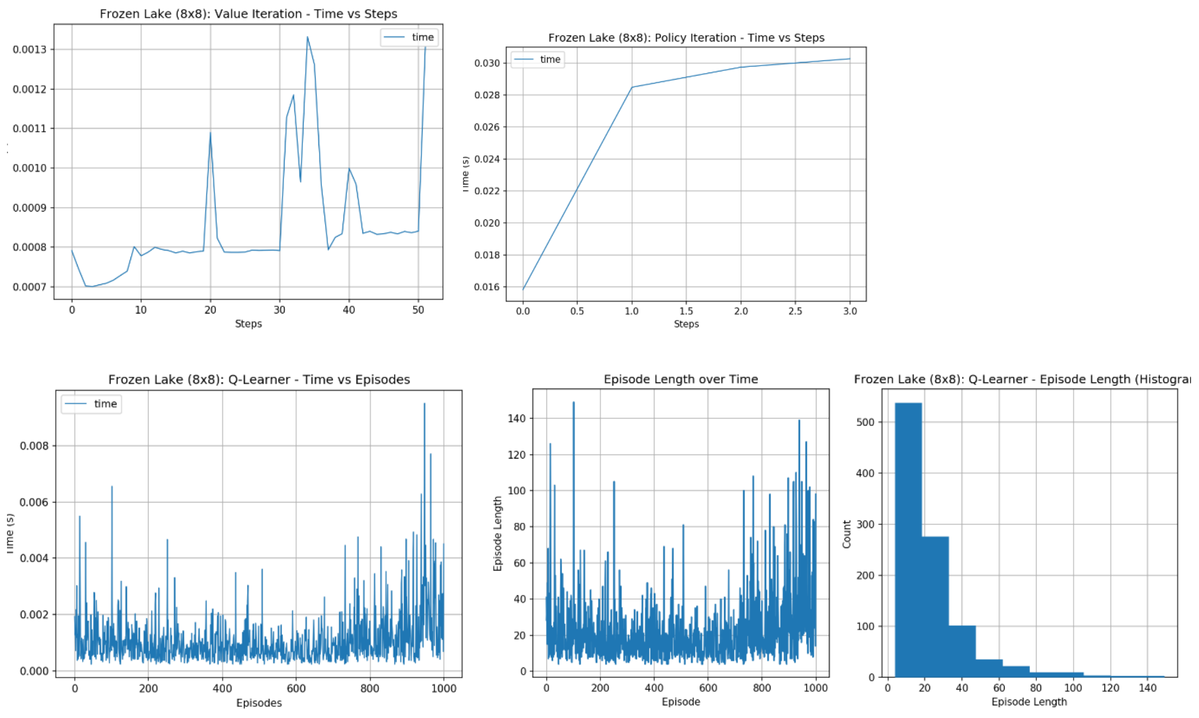


Figure 3: Iteration duration graphs of different algorithm and iteration length of Q learning

Since VI takes all states in consideration the duration of each iteration is increasing with the number of iterations, which also increases the utility of states to compute. PI however only considers the policy update instead of value, first few steps are computationally expensive. With each iteration the computation reduces as the possible number of actions reduces. With Q-learning the iterations have almost same average duration and it is fluctuating all the way to the 1000th iterations. Since it is model free method, the computation does not decrease. With different epsilon and gamma(0.99) we may reach the convergence. Due to the time constraints we could not do the experiment here.

One last note about the discount factor, with lower discount factor all the algorithm takes more steps to converge. It may be because it reduces the exploitation and increases the explorations. However the difference in total reward and number of steps are very low within the range of 1-3 steps for VI and PI, as it is a very small space problem.

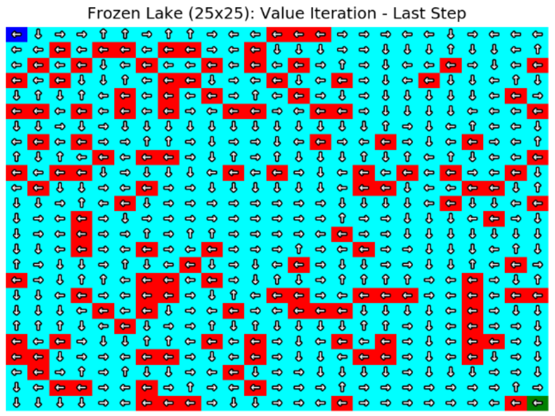
Large Lake(25x25) Analysis:

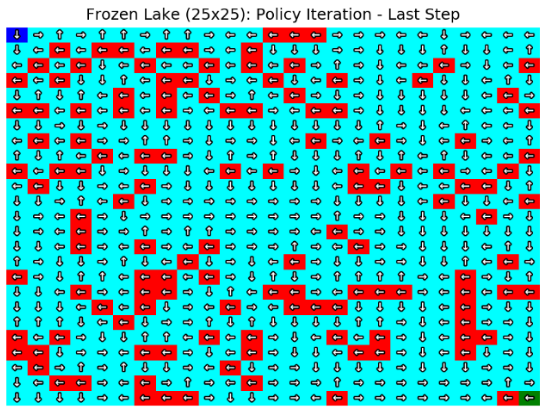
For the larger space problem, we used the following parameters for to reach the optimal policy:

	Max discount factor	Min discount factor	alpha	q_init	epsilon	epsilon_decay	Max Steps
VI	0.9	0.1					200
PI	0.9	0.1					200
Q	0.9	-	0.5	random	0.5	0.0001	1000

Table 1: Parameters used for the optimal policy for different algorithms

The followings are the states we found at optimal policy:





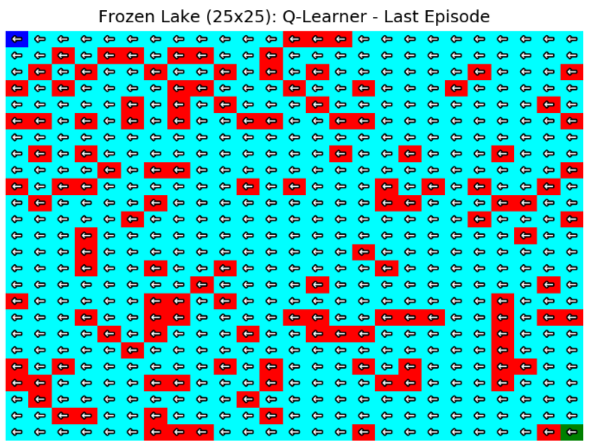
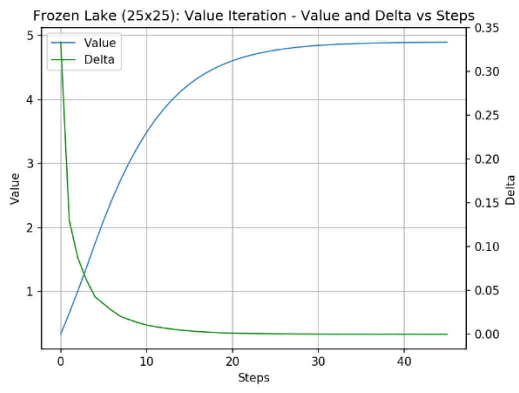


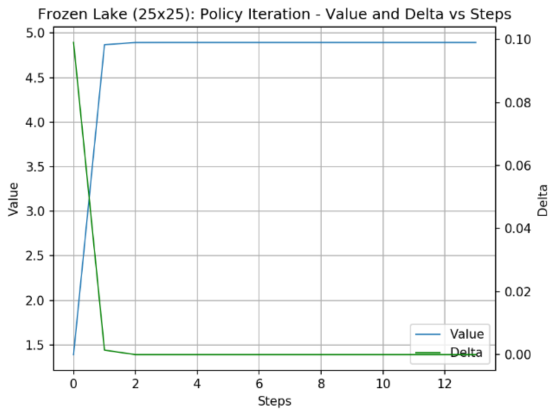
Figure 4: States at the last step of each algorithm after at convergence

The followings are the performance comparison of each algorithms:

Algorithms	Steps to Convergences	Time (s)	Converged	Reward
VI	56	0.33	YES	4.89
PI	14	3.18	YES	4.89
Q-learning	200	0.138	YES	4.89

Table 2: Performance comparison of different algorithms





Conclusion:

VI works better for both smaller and larger complex problems, even though PI outputs same policy. VI converges faster. Shorter steps of PI may look better but the time duration is higher. Although in this experiment Q-learning did not perform well due to the nature of the problem, however with more experiments with tuned parameters it would do better. In many real life problems Q-learning performs better. However, we left it for future experiments.

Work Cited:

[1] OpenAI Examples. https://github.com/openai/gym/blob/master/gym/envs/toy_text/frozen_lake.py

[2] https://en.wikipedia.org/wiki/Q-learning

[3] http://www.dudonwai.com/docs/gt-omscs-cs7641-a4.pdf?pdf=gt-omscs-cs7641-a4

[4] https://github.com/cmaron/CS-7641-assignments