CS7641 Assignment 4: Markov Decision Processes

Boyuan Liu
bryanliu@gatech.edu

1. Introduction

Reinforcement learning is one of the three main areas of machine learning, along with supervised learning and unsupervised learning. The Markov Decision Process (MDP) is a key concept in reinforcement learning. It provides a mathematical framework for decision-making in stochastic environments. In this assignment, I explored two MDP problems, the Frozen Lake and Forest Management, with reinforcement learning algorithms. I used model-based Value Iteration and Policy Iteration algorithms and a model-free Q-Learning algorithm. The experiments aim to study how various factors, such as state size and discount rate, influence the algorithms and their policy.

2. MDP Problems

The two MDP problems explored in this assignment are the Frozen Lake and the Forest problem. The Frozen Lake problem is a classic grid-world problem. The default configuration is made of a 4x4 grid that has a start state, a goal state, and several holes scattered randomly. The agent can take the action of up, down, left, and right at each state. The agent would receive a reward of 1 only when a goal state is reached. The game will restart if the agent reaches a hole. The Frozen Lake is an interesting MDP problem because its policy is easy to visualize, and the characteristics of the problem, such as the size of its state space and number of holes, can be customized to see their impact on the policy. In the Forest Management problem, the goal is maximizing the reward by either wait or cut the forest. The forest has a probability of catching on fire at each time step. The agent receives a reward of 1 and goes back to state 0 if it decides to cut the forest, and it receives a higher reward if it decides to wait and reach the final state. Similar to the frozen lake problem, we can vary the state size and the probability of fire and see how these factors impact the policy.

3. Value Iteration

The value iteration algorithm computes the policy by estimating the value of each state in the MDP problem. It updates the estimated values iteratively using the following equation:

$$V^*(s) = \max_a \sum_{s'} T(s, a, s')[R(s, a, s') + \gamma V^*(s')]$$

I first explored how the discount rate, γ, influences the value iteration algorithm in the frozen lake and forest management problem. The algorithm was tested with γ ranging from 0.6 to 0.95. As shown in Figure 1, the discount rate has a similar effect on both problems. A higher discount rate means future rewards are more important to the algorithm, which leads to the algorithm converging at higher reward values. This is because the value function is the estimated current reward plus the discounted future reward. A higher discount rate means a higher value of the discounted future rewards. A higher discount rate also leads to more iterations and computation time needed to converge.

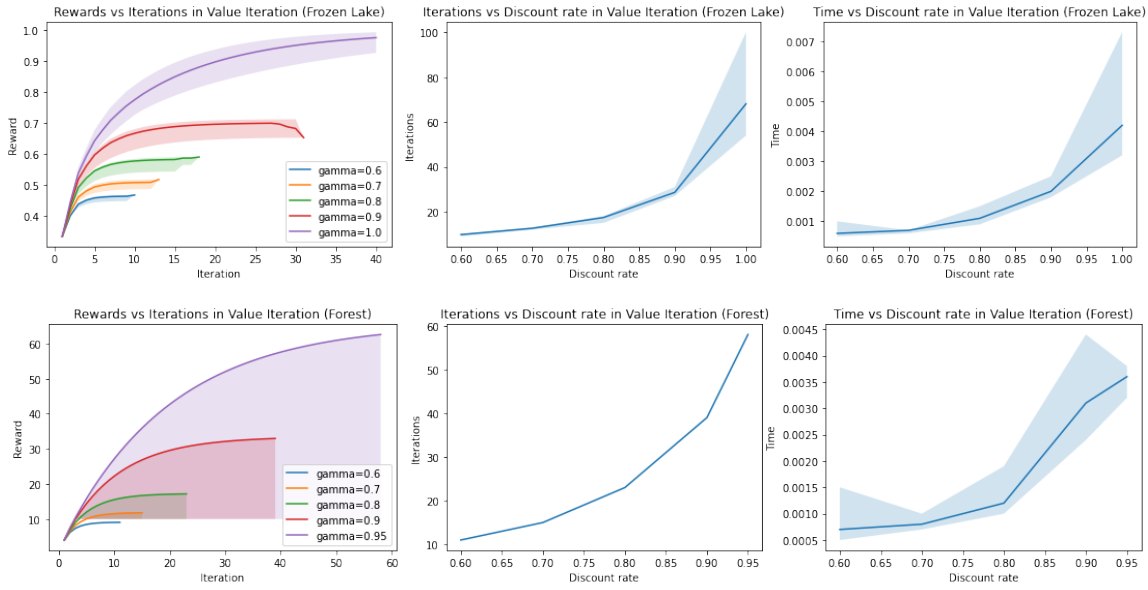

Figure 1 - Performance of the value iteration algorithm with different discount rates.

The policy generated by the value iteration algorithm for the forest management problem is also shown in Figure 2 below. The policies tend to cut the forest at the early stages since they are far away from the final reward. However, at later stages, the policies with higher discount rates start to preserve the forest earlier than the other policies. This observation makes sense because the policies with higher discount rates view future rewards more importantly.

```
Value Iteratiion Policy (gamma=0.95): (0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0)
Value Iteratiion Policy (gamma=0.9):  (0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0)
Value Iteratiion Policy (gamma=0.8):  (0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0)
Value Iteratiion Policy (gamma=0.7):  (0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0)
Value Iteratiion Policy (gamma=0.6):  (0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0)
```

Figure 2 - Policies generated by the value iteration algorithm with different discount rates in the forest management problem.

The value iteration algorithm uses an epsilon value to determine if the algorithm has converged. As shown in Figure 3, algorithms with higher epsilon values tend to require fewer iterations and computation time to converge. Although epsilon does not have a significant effect on the algorithm's performance, it is important to choose an optimal epsilon value to reduce computation time, especially in larger MDP problems.

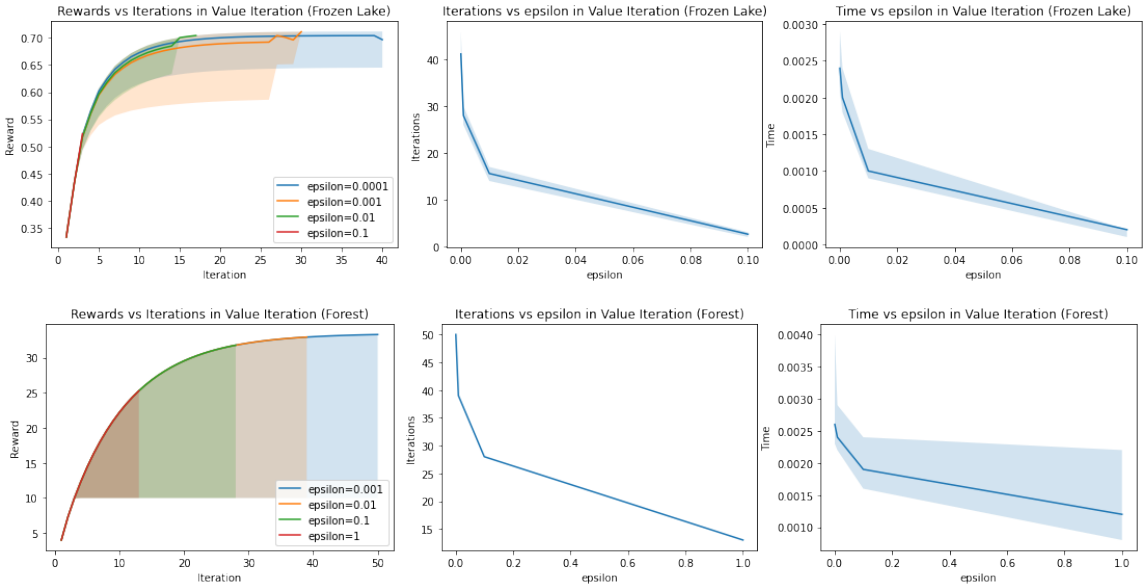

Figure 3 - Performance of the value iteration algorithm with different epsilon values.

State sizes sometimes have a significant impact on the performance of reinforcement learning algorithms. I experimented with different state sizes in both MDP problems, and the results are shown in Figure 4 below. In the frozen lake problem, the problems with different state sizes were able to converge at very close reward values. The state sizes did not appear to have a strong impact on the iterations required to converge. However, the computation time of each iteration grows exponentially with the state size. In the forest management problem, the problems with different state sizes converged at very close reward values, similar to the frozen lake problem. The state size did not have any impact

on the number of iterations required but the computation time of each iteration increased with the state size.

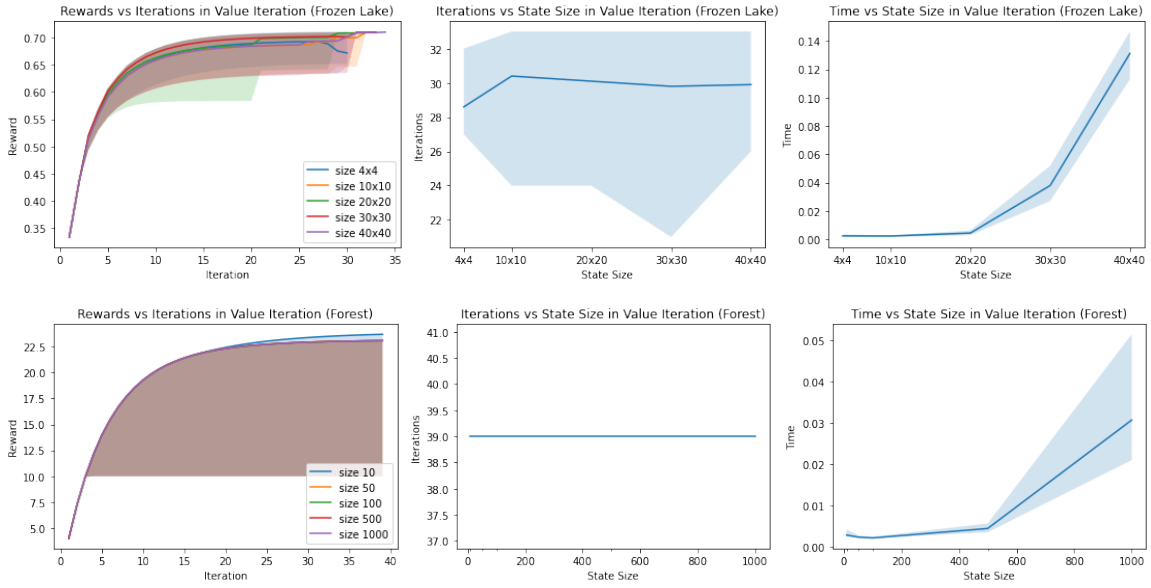

Figure 4 - Performance of the value iteration algorithm on problems with different state sizes.

Visualizations of the policies for the frozen lake problem are shown in Figure 5 below. The green and orange squares represent the start and finish state, while the blue and gray squares represent the frozen locations and the holes. The white arrows in the squares represent the directions generated according to the policy. As shown in the figure, the frozen squares around the holes all have arrows pointing away from the hole, and the arrows in the rest of the frozen squares generally point towards the finish state. This indicates that the agent has learned to move towards the finish state while avoiding the holes.

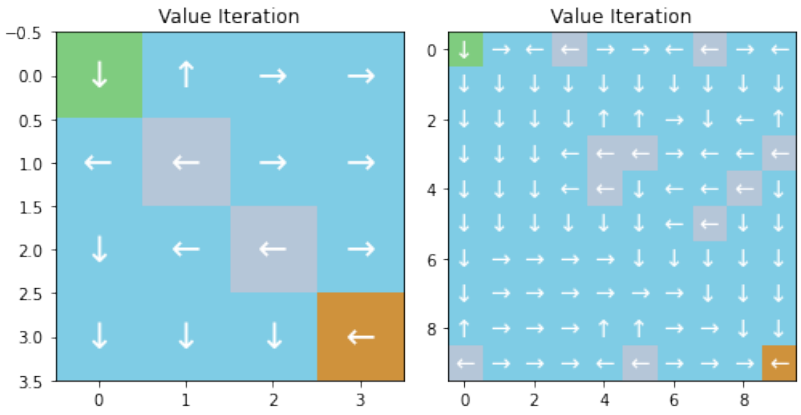

Figure 5 - Policies generated by the value iteration algorithm on frozen lake problems with different state sizes.

4. Policy Iteration

The policy iteration algorithm uses an iterative method similar to the value iteration algorithm. However, instead of updating the value function iteratively, it uses a two-step approach shown in the equations below. It updates the policy π after each value iteration. This two-step mechanism often leads to fewer iterations required to converge.

$$V^{\pi}(s) = R(s,\pi(s)) + \gamma \sum_{s' \in S} T(s,\pi(s),s') V^{\pi}(s')$$

$$\pi'(s) = \arg\max_{a} \left\{ R(s,a) + \gamma \sum_{s' \in S} T(s,a,s') V^{\pi}(s') \right\}$$

I first experimented with different discount rates and examined their impact on the algorithm's performance. Similar to the value iteration experiment, a higher discount rate led to the algorithm converging at higher reward values in both MDP problems. Higher discount rates also led to more iterations needed to converse, which was also observed in the value iteration experiment. The policy iteration algorithm converged within 10 iterations in both MDP problems. This is significantly less than the iterations required in the value iteration algorithm. In terms of overall computation time, the time consumption of policy iteration did not grow exponentially with the discount rate, and it was less than the time consumption of value iteration in general.

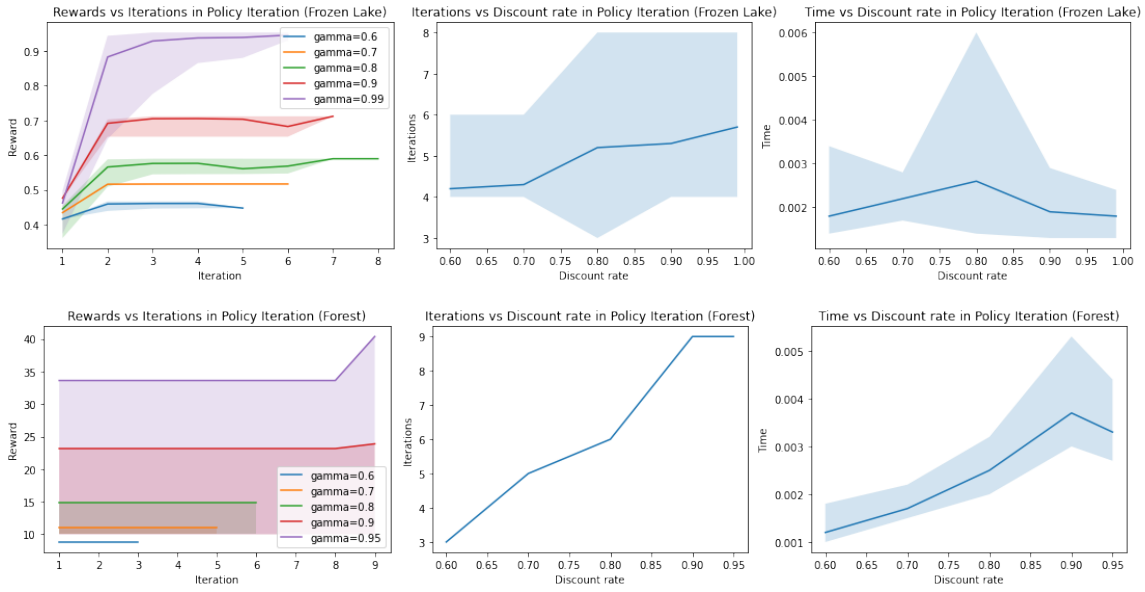

Figure 6 - Performance of the policy iteration algorithm with different discount rates

The forest management policies generated by policy iterations with different discount rates are shown in Figure 7 below. They are almost identical to the policies generated by value iterations. The policies tend to cut the forest at the early stages since they are far away from the final reward. At later stages, the policies with higher discount rates start to preserve the forest earlier than the other policies. This observation showed that the policies with higher discount rates view future rewards more importantly.

```
Policy Iteratiion Policy (gamma=0.95): (0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0)
Policy Iteratiion Policy (gamma=0.9):  (0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0)
Policy Iteratiion Policy (gamma=0.8):  (0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0)
Policy Iteratiion Policy (gamma=0.7):  (0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0)
Policy Iteratiion Policy (gamma=0.6):  (0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0)
```

Figure 7 - Policies generated by the policy iteration algorithm with different discount rates in the forest management problem.

The performance of the policy iteration algorithm on different state sizes is shown in Figure 8. Unlike in the value iteration experiments, the state sizes appear to have a noticeable influence on the number of iterations needed to converge. Problems with larger state sizes generally require more iterations to converge, and their computation time also grows exponentially with the state sizes.

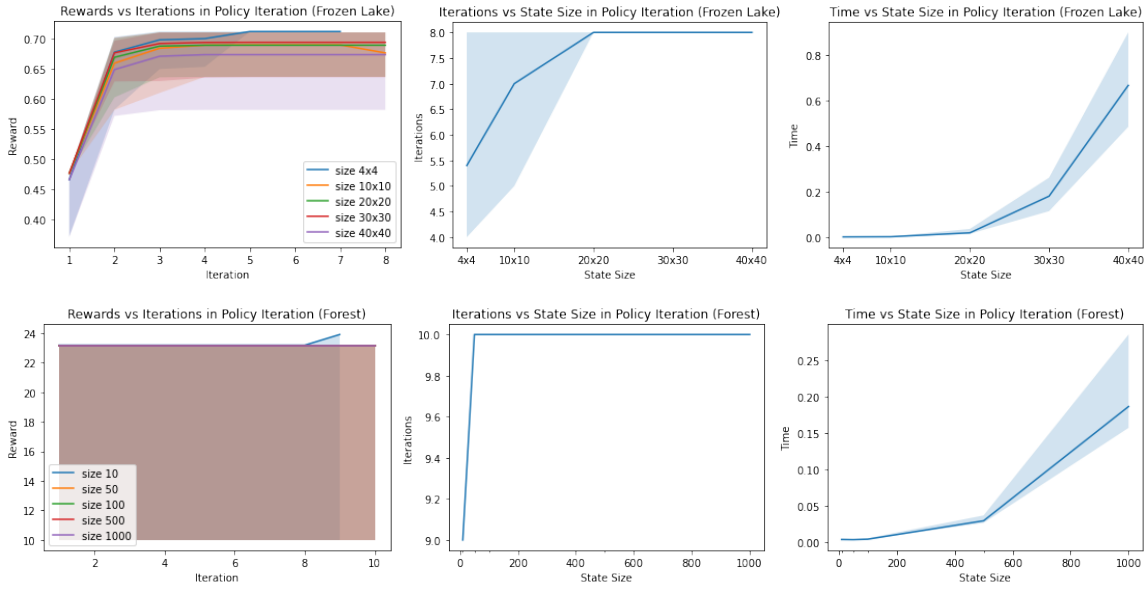

Figure 8 - Performance of the policy iteration algorithm on problems with different state sizes

Visualizations of the policies for the frozen lake problem are shown in Figure 9 below. The policy map is illustrated in the same way as Figure 5. As shown in the figure, the frozen squares around the holes all have arrows pointing away from the hole, while the

arrows in the rest of the frozen squares generally point toward the finish state. This indicates that the agent has learned to move toward the finish state while avoiding the holes. These policies are very close to the policies generated by value iteration, with a few minor differences. The value iteration and policy iteration algorithms converged to very similar policies at different state sizes.

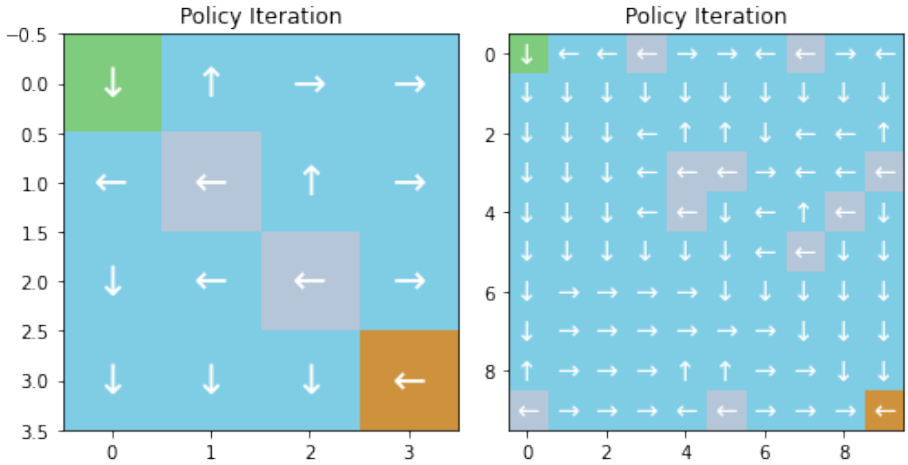

Figure 9 - Policies generated by the policy iteration algorithm on frozen lake problems with different state sizes.

5. Q-Learning

Unlike the model-based value iteration and policy iteration algorithms, the Q-learning algorithm is a model-free reinforcement learning algorithm. It uses the following equation to calculate the Q-values of each state-action pair and generates policies based on the Q-values. Model-free reinforcement learning algorithms do not require probability matrices. It has broader applications than model-based algorithms because the probability matrices are often hard to obtain in real-world applications.

$$Q(s,a) = r(s,a) + \gamma \max_{a} Q(s',a)$$

The Q-learning algorithm requires much more iterations to converge than policy iteration and value iteration due to its large Q-tables. The algorithm is considered converged when the Q-value functions no longer change. The average Q-values versus iterations are shown in Figure 10. The Q-values were initialized at zero. In the frozen lake problem, there was a sharp increase in average Q-values at the beginning as the algorithm began learning. The Q-values then began to change less and eventually converged after 4e6 iterations. The average Q-values in the forest management problem showed a very similar

trend. The Q-learning algorithm took 212 seconds to converge in the 4x4 frozen lake problem and 393 seconds to converge in the 20-state forest management problem.

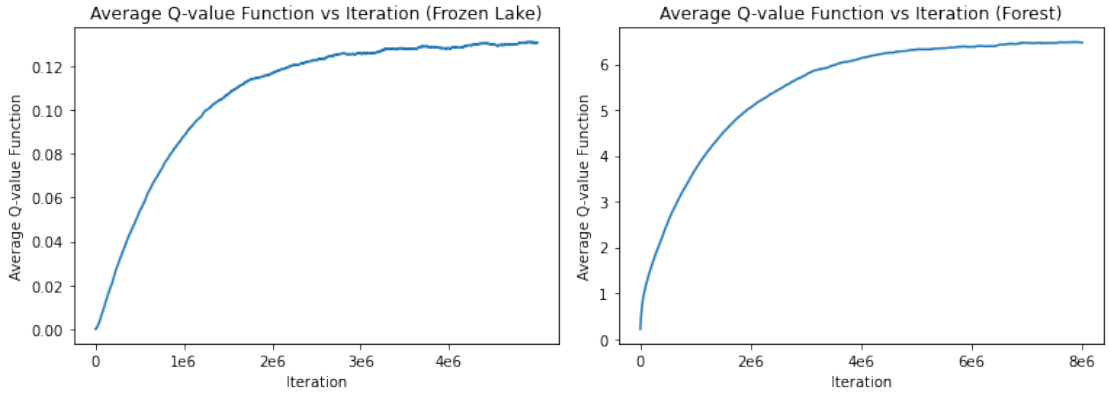

Figure 10 - Convergence plot of the Q-learning algorithm on the frozen lake and forest management problem.

The policies generated by all three algorithms on the 4x4 frozen lake and 20-state forest management problems are shown in Figure 12 and Figure 13. In the frozen lake problem, the policies generated by value iteration and policy iteration are very similar. There is only one difference at location (2, 3). The policy generated by the Q-learning algorithm has several differences. These differences appear to be non-optimal compared to the other two policies. In the forest management problem, the value iteration and policy iteration algorithms generate the same policy, while the Q-learning algorithm tends to cut the forest at later stages, which is not ideal considering the higher final reward. Overall, value iteration and policy iteration both performed very well on the two MDP problems. The policies generated by the Q-learning algorithm appear to be non-optimal, and further tuning is required.

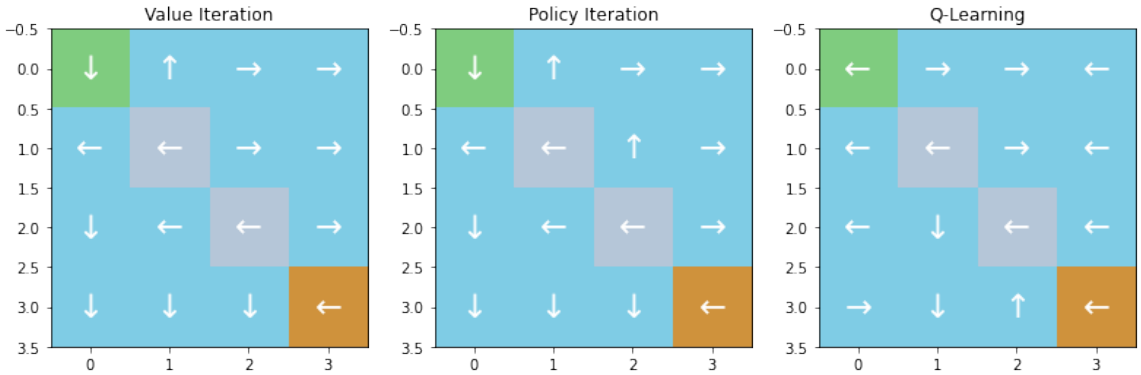

Figure 12 - Policies generated by the value iteration, policy iteration, and Q-learning algorithm on the 4x4 frozen lake problem.

```
Value Iteratiion Policy:   (0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0)
Policy Iteratiion Policy:  (0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0)
Q-Learning Policy:         (0, 1, 1, 0, 0, 1, 1, 0, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 0, 0)
```

Figure 13 - Policies generated by the value iteration, policy iteration, and Q-learning algorithm on the 20-state forest management problem.

6. Conclusion

In this assignment, several experiments were conducted to explore the value iteration, policy iteration, and Q-learning algorithm and to examine their behaviors on the frozen lake and forest management problem. Factors such as discount rate, epsilon, and state size have shown different effects on the algorithms. Value iteration and policy iteration had similar performances, and the policies generated by the two algorithms are also very similar. The Q-learning algorithm did not perform as well as value iteration and policy iteration. Further tuning of parameters such as exploration/exploitation, learning rate, and the discount rate is required to further improve its performance and reduce computation time.