Markov Decision Processes
CS 7641

Josh Adams
jadams334@gatech.edu

1 INTRODUCTION

This assignment will compare Value Iteration (VI), Policy Iteration (PI) and Q Learning for solving Markov Decision Process (MDP) type of problems. VI consists of finding the optimal value function for a problem, once a value function is considered optimal, the resulting policy is also considered to be optimal. VI is considered converged when the value function at time t is the same as the value function at t+1. PI consists of starting with a random policy and iteratively evaluating and improving the policy until the policy converges. PI is considered converged when the policy as time t is the same as at time t+1. Q learner uses a Q table to provide best actions to take at a given state. I considered the Q learner to be converged when the resulting Q table and expected reward has not changed for some number of iterations. There will be a small MDP and a large MDP, the first is called the frozen lake of size 10x10 which is a grid-world type of environment and the second is called forest management which is not considered a grid-world type of environment. The frozen lake will be considered the small problem as it will consists of only 100 states and the forest will be the large and will contain 1000 states. I will compare the two problems and show that number of states is not indicative of the difficulty. The frozen lake problem while only having 100 states is much more complicated than the forest management problem with 1000.

2 Q LEARNER

I have modified the Q learner to be try and maximize exploitation and exploration. Exploitation is when the Q learner decides to take the best action defined in its q table based on its current state. Exploration is where the Q learner takes a random action in the hopes it provides higher rewards later in the problem. The parameter epsilon determines the rate at which the Q learner explores or exploits. My Q learner checks to see if a randomly generated number is larger than epsilon, if it is larger, then a random action is taken. I inversely decay the epsilon by a rate for each iteration, this increases the epsilon over time and reduces exploration. The Q learner in the beginning will explore much more and exploit the knowledge it has gained later. I found that the starting epsilon value as well as the decay rate play an important role in the performance of the learner. A starting epsilon value of 0.001 and a decay rate of 0.00001 performed well but not as well as policy iteration so I implemented another feature which resets the epsilon after a certain number of convergence checks have occurred. This allowed my Q learner to start exploring again later in the problem and helped my Q learner to perform just as well as VI and PI.

3 FROZEN LAKE

The frozen lake problem consists of a large grid type environment where each space can be one of four states. There is the starting state which is there the problem starts. The goal state which indicates the problem is over as the goal has been reached. There are also two other states which dramatically influence the way the environment is solved. The first is a frozen state, this is a relatively safe state because you can continue solving the problem. The last is a terminating state which is indicative of a hole in the lake, where you fall into the water and end the experiment. What makes this problem interesting is that our agent is attempting to navigate this environment and make it to the goal state without falling into a hole and having to start over as well as trying to maximize our rewards. Each of the algorithms have various hyperparameters, such as gamma which is the discount rate and epsilon which is typically used for exploring the environment.

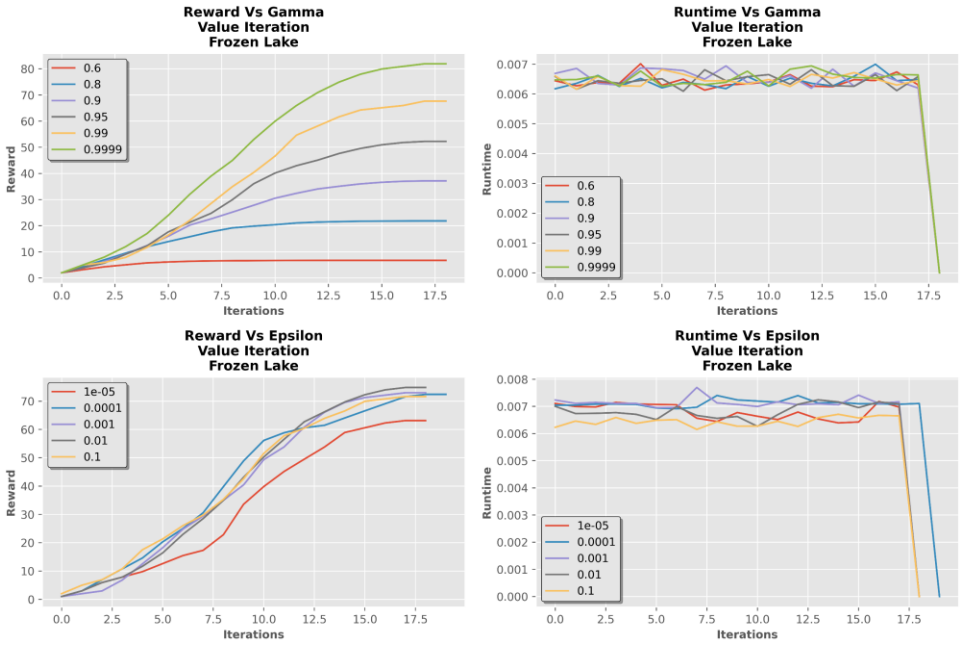

Figure 1 — Value Iteration with different hyperparameters on Frozen Lake.

For VI gamma was the most impactful regarding the reward generated by the policy. The reason I believe this is happening is because as gamma gets lower, the algorithm values the current action less and this may lead to making inefficient actions subsequently ending at a suboptimal policy. Neither hyperparameter changed the convergence much, all converged after around 17 iterations.

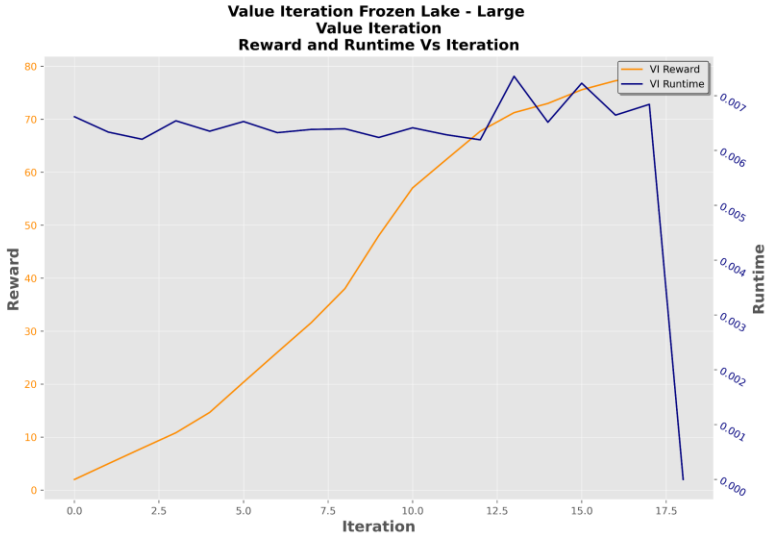

Figure 2 — Value Iteration Runtime and Reward Vs Iteration on Frozen Lake.

Looking at figure 2, you see that the runtime is consistent until getting close to convergence. I would expect this was due to iterating over more values to find the best policy. There could be many possible values to use which still produce good results thus it must test many more values closer to the convergence point than when starting out.

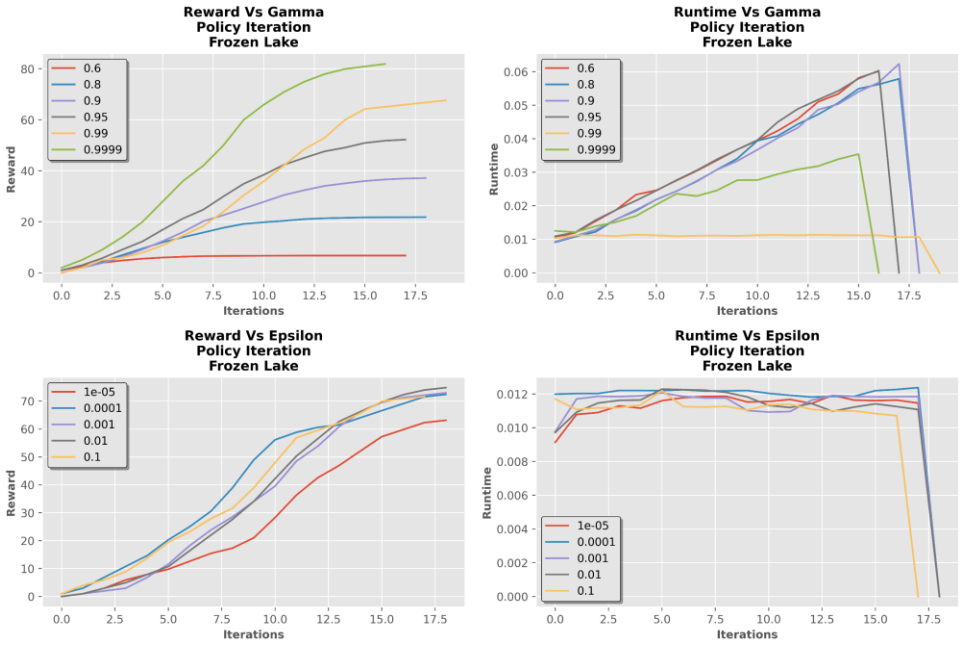

Figure 3 — Policy Iteration with different hyperparameters on Frozen Lake.

Figure 3 shows some interesting points such as varying the gamma did change the runtime and the point of convergence. When the gamma was very high at 0.9999, PI was able to converge faster than other values of gamma at around 15 iterations, but I was not expecting the highest gamma to also produce relatively high runtimes. My expectation was the higher the gamma the lower the runtime.

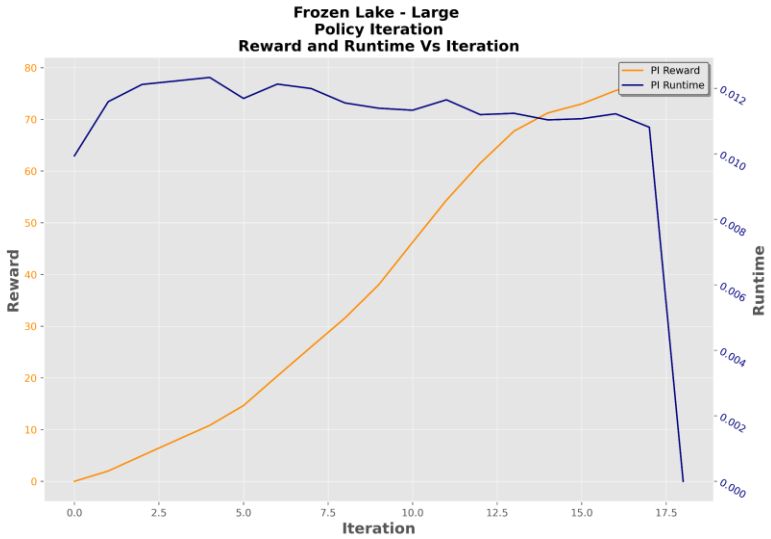

Figure 4 — Policy Iteration Runtime and Reward Vs Iteration on Frozen Lake.

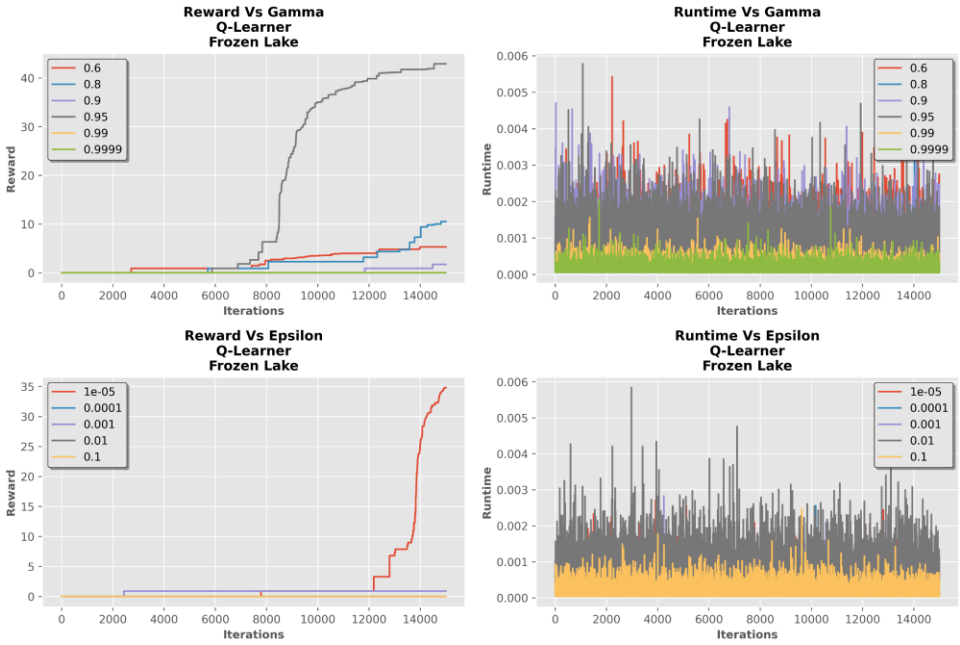

Figure 5— Q Learner with different hyperparameters on Frozen Lake.

Figure 5 shows the changes that epsilon and gamma had on the Q learner; both impact the results in different ways. The changes caused by different epsilon values were more dramatic, such as a 0.01 epsilon produced about a 1 reward while an epsilon of 0.00001 produced around 35. The reason this is happening is due to how I have epsilon set up in the Q learner. The lower the epsilon is, then a random action will be chosen more frequently —in my implementation of Q learner— . Looking at the chart for reward vs gamma it shows that a gamma of 0.95 was much better than the other values. I would expect that as gamma got smaller the reward generated would happen sooner in the graph. A gamma of 0.99 may get to some N reward after 1000 iterations and if gamma was set to 0.75 that same reward may take tens of thousands to achieve. There is a noticeable difference between the runtimes of the gamma varied graphs and the epsilon varied. A high gamma reduced the runtimes because the immediate reward was more significant thus was chosen and allowed for less exploration.

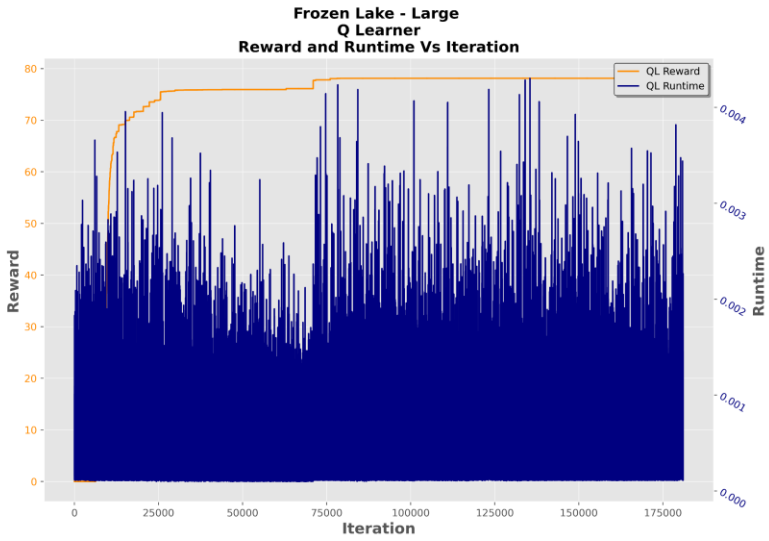

Figure 6— Q Learner Runtime and Reward Vs Iteration on Frozen Lake.

Figure 6 is one of the most interesting graphs because not only does it show that the runtimes varied quite a lot but within the runtime you can distinguish a pattern which corresponds to the way I implemented the Q learner. The runtime starts off a little high and then is continually lowering to around 75000 iterations and then jumps back up. You can also see this jump with the reward around the

same time. This is happening because for each iteration I inversely decay the epsilon with respect to the iteration and convergence. Once a converging point is detected a counter is incremented each iteration with no improvements. Once a certain value for this counter is reached, I reset epsilon and allow for the algorithm to begin searching the environment again, like it did at the beginning. The Q learner was able to get further rewards because it could explore versus exploit the environment. I believe I would have been able to get further rewards with higher number of iterations and further optimization of hyperparameters. Both VI and PI were able to find optimal policies extremely quickly both were within less than 1 second and both found the same optimal policy. All three policies were similar, but the Q learners' policy was different than VI and PI. The Q learner took over one hundred seconds to produce its optimal policy.

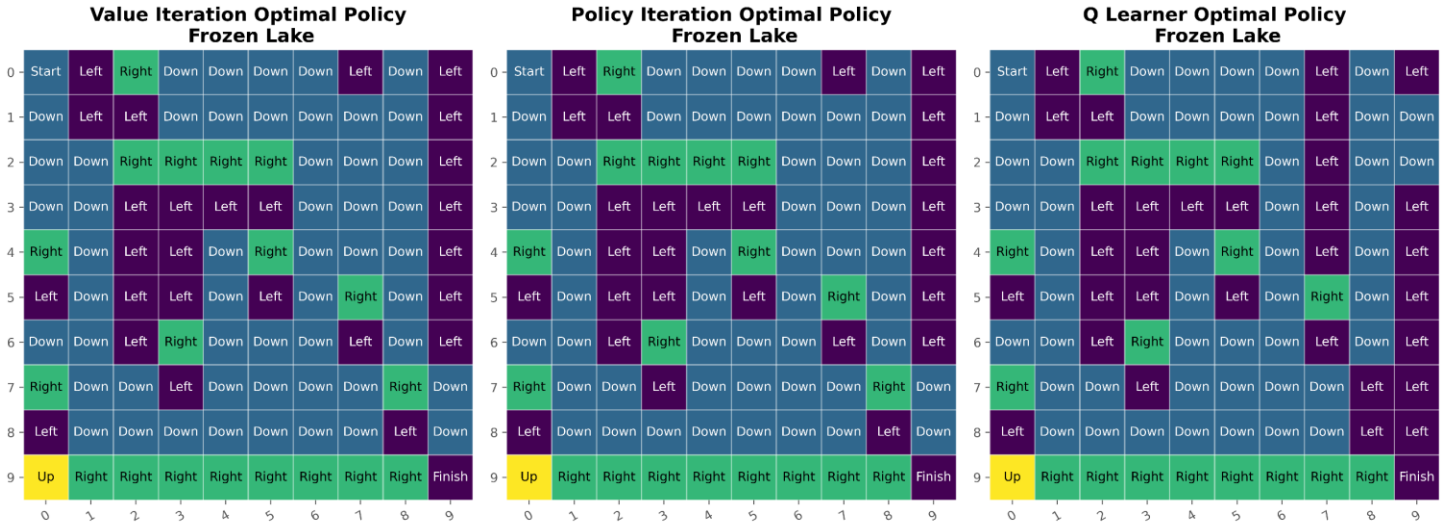

Figure 7— Optimal policies visualized for VI, PI and Q Learner on Frozen Lake.

In terms of number of iterations, both VI with 17 iterations and PI with 19 iterations are far superior to my implementation of Q learner with 399999, which was the limit I set for total iterations. One measurement the Q learner was able to outperform the other algorithms was with number of evaluation games won. This is where we evaluate the policy on an environment and track how many we win. Both VI and PI performed identically only able to win 843/1000 games, while the Q learners' policy was able to win every game it played. As the number of states increase both VI and PI perform relatively the same, but it becomes apparent my implementation of Q learner is falling behind.

Table 1 — Results for the frozen lake MDP with VI, PI and QL.

Name	Number of Iterations	Total Runtime	Highest Reward	Evaluation Games Won
Value Iteration	17	0.14515s	74.764	843/1000
Policy Iteration	19	0.21999s	74.764	843/1000
Q-Learner	329303	166.23939s	73.361	1000/1000

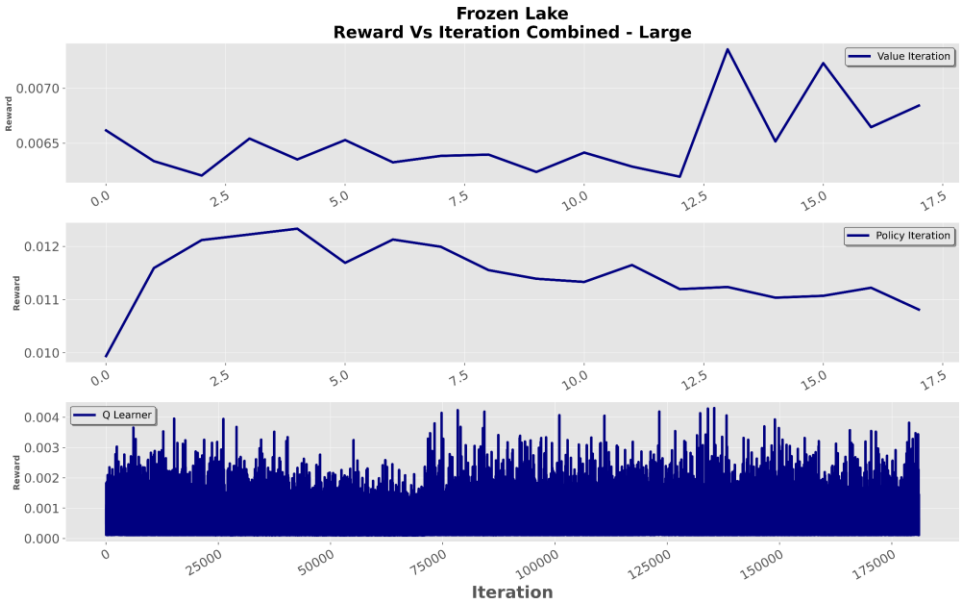

Figure 8— Reward per iteration of VI, PI and Q learner on Frozen Lake.

3.1 Frozen Lake Conclusion

In conclusion both VI and PI performed similarly in the Frozen Lake environment. VI was able to converge on an optimal policy in less time than PI and Q learning as well as in less iterations. VI took 17 iterations and 0.14515 seconds to converge, PI took 19 iterations at 0.219999 seconds and Q learning came in last in terms of runtime and number of iterations because it took Q learning 329303 iterations and 166.23939 seconds to converge. Both VI and PI converged to the same policy, which was expected, the Q learner found a different policy which was arguably better then both VI and PI. The policies from VI and PI managed to win 843 of the 1000 games tested while the Q learners' policy was able to win all 1000 games. To claim one algorithm is superior to another would be naïve as speed may be of concern then VI would be best, if evaluation accuracy were the highest priority then Q learning would be. It is subjective to the task and requirements.

4 FOREST MANAGEMENT

The second MDP is called forest management where the main goal is to maintain a forest and the secondary goal is to cut the trees to sell. Each year the forest has a 10% chance of catching fire and resetting the problem. What makes this problem so interesting is having multiple goals, maintaining the forest, and selling the cut trees, but also the chance to have it reset by the fire. The forest management MDP will have 1000 states and be considered the large problem in this assignment.

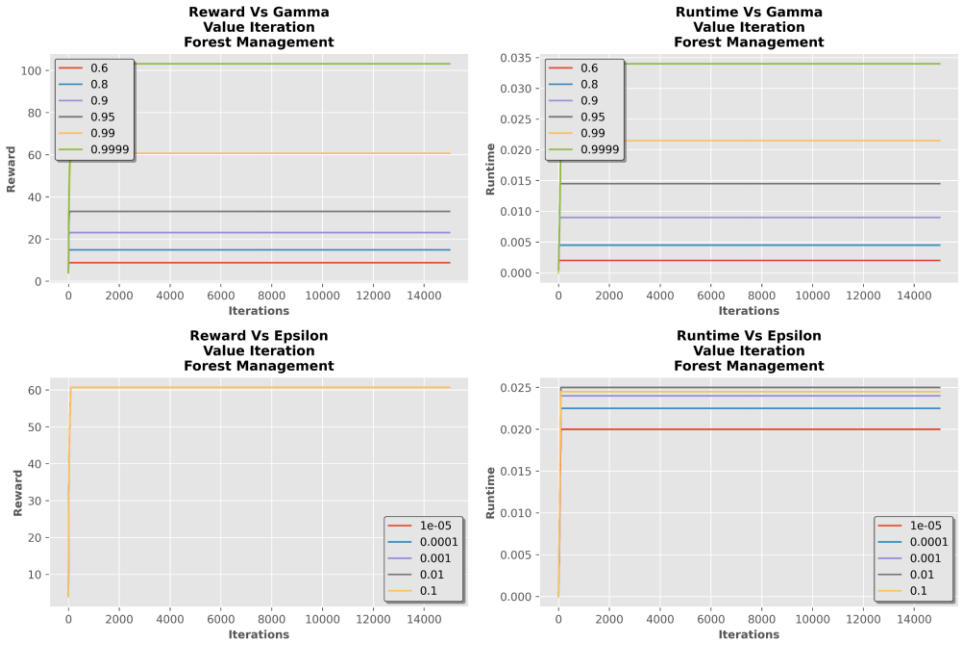

Figure 9 — Value Iteration with different hyperparameters on Forest Management.

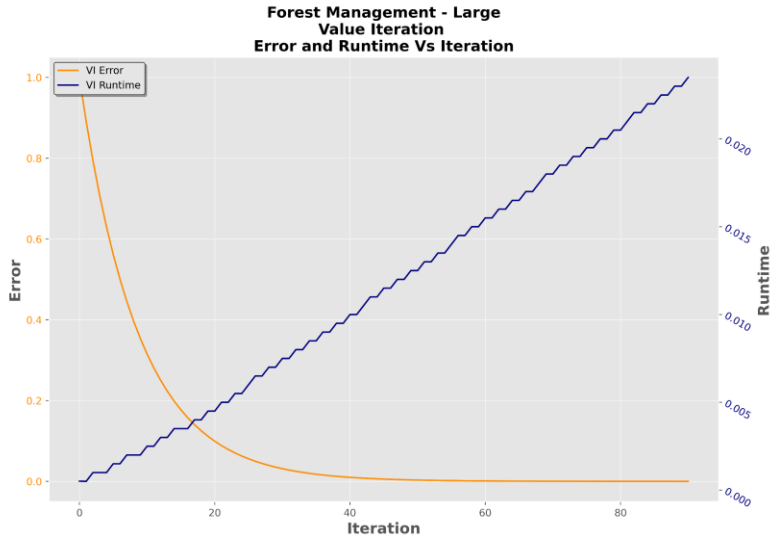

Figure 10 — Q Learner Runtime and Reward Vs Iteration on Forest Management

Figure 9 shows the impact that varying gamma and epsilon had on VI for the forest management problem. Both reward and runtime seem to be corelated. A low gamma produced low rewards but as well as a low runtime, conversely a high gamma produces better rewards but also increases the runtime. Epsilon had no impact on reward but did change the runtime. Figure 10 shows the error curve along with the run time, the step like pattern in the run time is due to the iterative nature of VI.

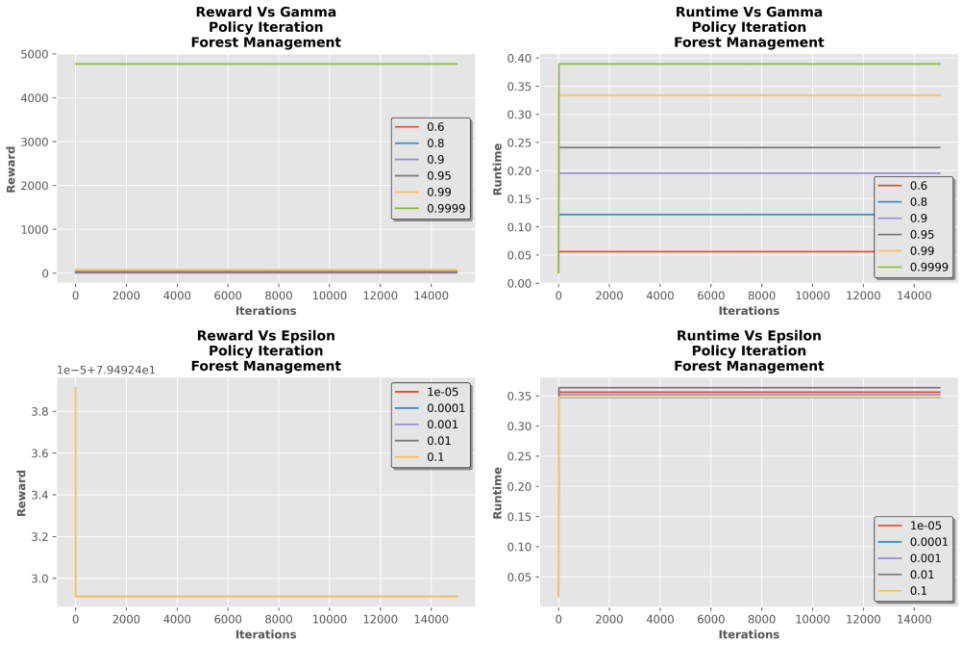

Figure 11 — Policy Iteration with different hyperparameters on Forest Management.

I am not exactly sure what is happening in Figure 11. I ran the experiments three times to verify and each was very similar to this result. I can hypothesis what is happening, but I am not confident it will be accurate. Gamma's impact on reward was not apparent to me but the changes it made to runtime, do. When gamma is high it means the algorithm values current rewards more than later rewards. This would suggest when it found a good policy it was more likely to follow it versus trying other actions. This allows the algorithm to run longer and I believe the case here was that the algorithms were not finding solutions and ended per some other criteria. This would explain the reward versus gamma and having found an actual solution, but the rewards value was incredibly high —I assume due to an error—.

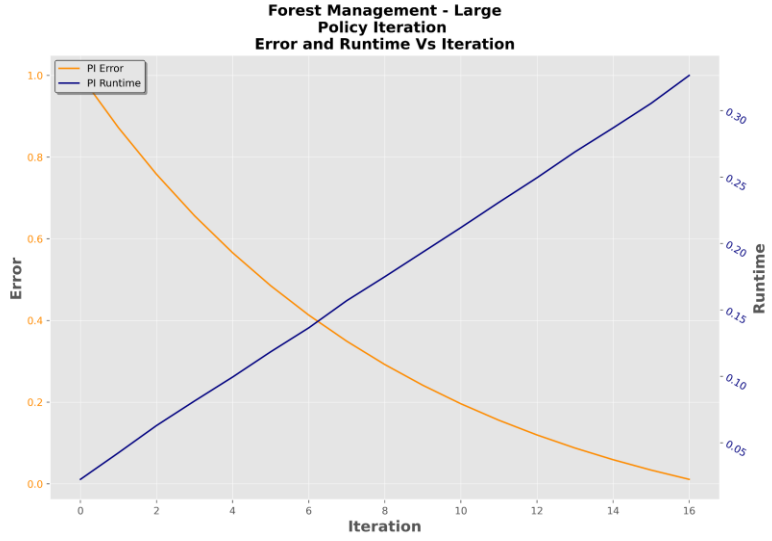

Figure 12 — Policy Iteration Runtime and Reward Vs Iteration on Forest Management.

Figure 11 is very consistent with figure 9 and I would expect this because of how VI and PI process the environment in similar fashions.

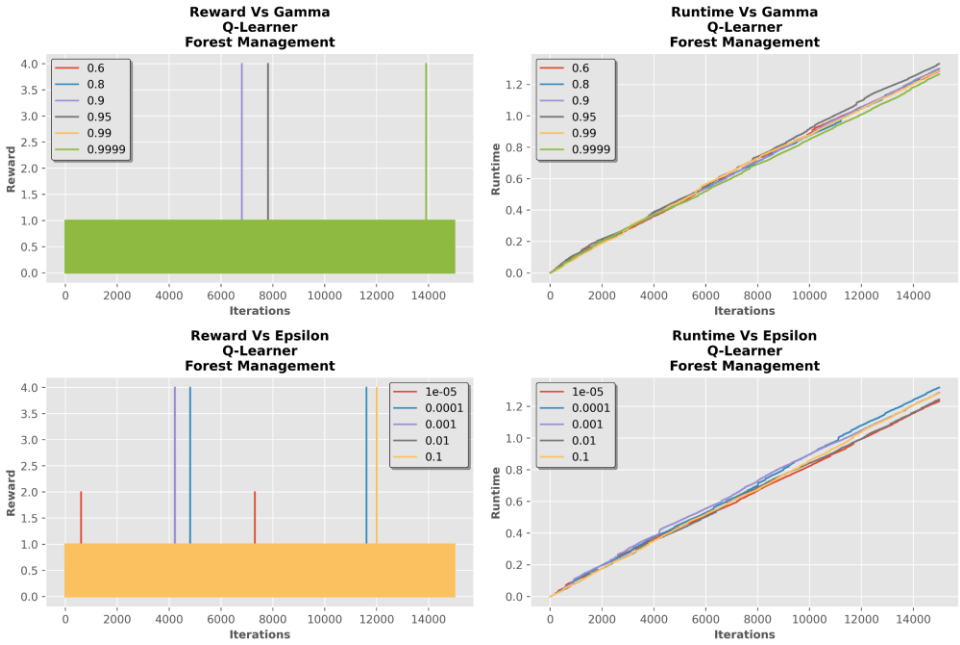

Figure 13 — Q Learner with different hyperparameters on Forest Management.

The hyperparameters gamma and epsilon control much of the behavior of the Q learner. Gamma being the discount rate controls how much an immediate reward is valued versus one in the future. Gamma did not make much of a difference in the overall reward. Epsilon did make a difference, albeit not a very impactful one. Epsilon allowed for the learner to get a reward of four, more times than with gamma but also allowed for partial rewards. Neither hyperparameter changed the runtimes much.

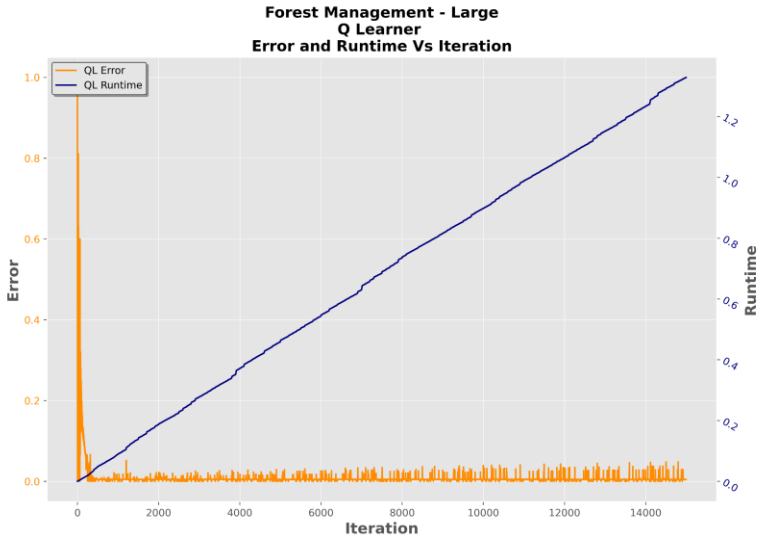

Figure 14 — Q Learner Runtime and Reward Vs Iteration on Forest Management.

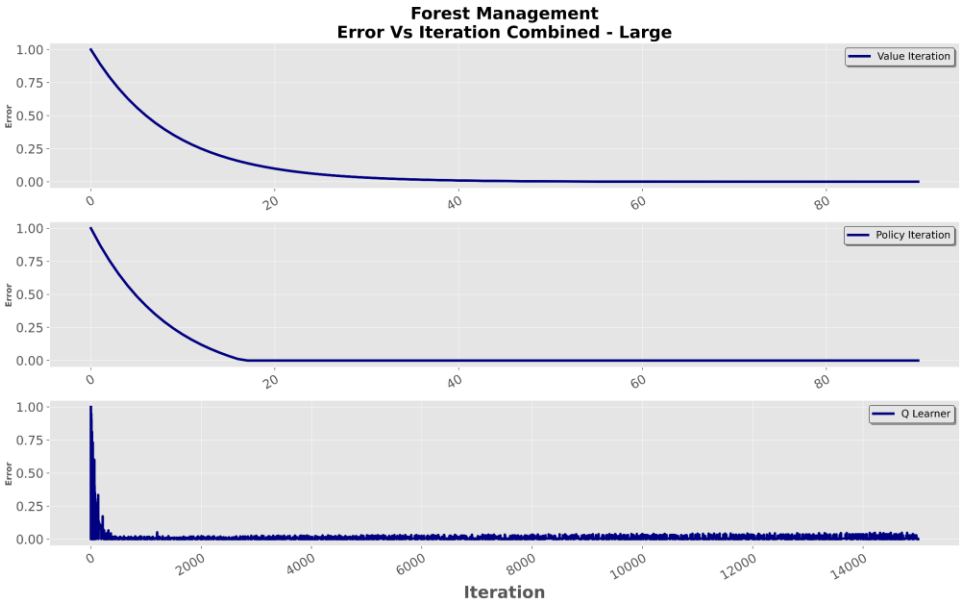

Figure 15— Error Vs Iteration Forest Management.

4.1 Forest Management Conclusion

VI, PI and the Q learner performed similarly in terms of runtime. The biggest difference between these three algorithms is number of iterations to converge. Like in the frozen lake MDP, VI and PI were able to converge in many less iterations than Q learning. Ultimately VI took around 50 iterations to converge, PI took less than 20 and Q learning took over 10,000 iterations. These algorithms performed differently in the two environments. In Frozen lake, VI was able to converge the fastest and in the fewest iterations while in this Forest management PI was able to beat VI. I expect the reason is that in Forest management there are many more states which would cause VI to have to iterate over many more values, thus increasing its run time and iterations to converge. Another difference is that Q learning was able to converge in less than ten seconds while in the Frozen lake MDP it took over 150 seconds.

5 REFERENCES

1. https://github.com/dennybritz/reinforcement-learning/blob/master/DP/Policy%20Iteration%20Solution.ipynb
2. https://github.com/dennybritz/reinforcement-learning/blob/master/DP/Value%20Iteration%20Solution.ipynb
3. https://github.com/chaitanyamittal/Markov-Decision-Processes/blob/master/MDPs.ipynb
4. https://github.com/PacktPublishing/Reinforcement-Learning-Algorithms-with-Python/blob/master/Chapter03/frozenlake8x8_policyiteration.py
5. https://github.com/PacktPublishing/Reinforcement-Learning-Algorithms-with-Python/blob/master/Chapter03/frozenlake8x8_valueiteration.py
6. https://gym.openai.com/envs/FrozenLake-v0/
7. https://artint.info/2e/html/ArtInt2e.Ch9.S5.SS2.html
8. https://artint.info/2e/html/ArtInt2e.Ch9.S5.SS3.html
9. https://1drv.ms/u/s!AvQKh521Q29JhNcK6Vuz82Lqkftcyg?e=I6Icsm
10. https://www.dropbox.com/s/1ct1eg6siw3ey1w/MDP_Container.zip?dl=0