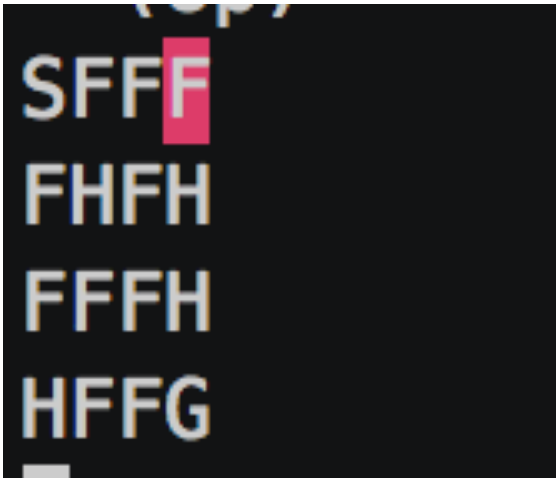
Rohith Krishnan
CS4641

Reinforcement Learning Analysis

Markov Decision Processes

Markov Decision Processes are discrete, stochastic environments that are used to model some process and provide a framework for decision making. Here we look at two different Markov Decision Processes that vary in terms of state space size, reward functions, and their respective transition models. These MDPs are from OpenAI which is a nonprofit research company for developing General Artificial Intelligence.

Frozen Lake. This MDP is a grid world where some of the tiles of the grid are safe to walk on and other tiles are hazards that lead the agent to fall into the lake. There is a single goal that the agent must get to and this goal has a reward of 2.5 while the hazard states have a reward of -0.5. All other states on the map have a reward of zero. The transitions that this agent takes in the environment are not deterministic as the agent has an equal probability in going in the three opposite directions than the one it took (i.e. if the agent decides to go left, it has a 33% chance of going up, down, or right). This environment has 16 total states and 4 possible actions (U, D, L, R). The map of this environment is shown below.



(Fig. 1: Frozen Lake World, S – start state, F – safe grid, H – hazard , G – Goal)

Taxi. This MDP is a grid world with a much larger state space than frozen lake as it has around 500 possible states in the environment. The goal for the agent is to pick up a passenger at one of the four special locations in the map and drop the passenger off at another location. The reward function gives 20 points for a successful drop off and subtracts one point for every time step it takes the agent to drop off the passenger. There is also a 10-point penalty for illegal pickup and drop-off actions taken by the agent. Unlike the frozen lake problem, the agent's actions are deterministic and the action taken will always be the one that occurs. The map of this world is shown below. The letters R, G, Y, and B are the different locations that the agent must pick-up

and drop-off passengers to. The agent itself is represented by the yellow rectangle in the figure below.

(Fig 2. Taxi Grid World)

Why These Problems are Interesting

I found the Frozen Lake problem interesting because of the non-determinism of the actions that the agent takes in the environment. I was interested in seeing how whether Value Iteration or Policy Iteration gave the better policy to safely navigate the environment to reach the goal state. I also wanted to see how changing the reward function impacted the policies that these two techniques produced and if introducing higher negative reward for the hazards and higher positive reward for the goals changed the number of iterations it took for each algorithm to converge. I also tested each algorithm with different values of gamma to see how discounting the future reward of a state by different amounts changed the optimal policy produced by the two algorithms.

The Taxi problem is interesting because it models a real-life process and could have real world applications. Having a good reinforcement learning model of a real-life process can help in the development of autonomous systems for Taxi services. If an agent in this environment can learn the right idea of how to pick up and drop off passengers in a timely matter and at the correct locations then this model could possibly be used in the real world. Another point of interest for this problem is the size of the state space as there are 500 total states in the environment. I was interested in seeing how this large environment impacted the running time of the two algorithms. Unlike the Frozen Lake problem, all the states other than the goal state have a negative reward associated with it which introduces a notion of time. This means an optimal policy should cause the agent to get to the goal states in the shortest number of steps possible.

Value Iteration

The first algorithm I used to solve these Markov Decision Processes was Value Iteration. Value Iteration iterates through all the states in the state space of the environment and tries to maximize the utility of each state. The utility of a state is defined by the Bellman equation.

$$U^{\pi}(s) = \max_a \sum_{s'} T(s,a,s')\,(R(s,a,s') + \gamma U^{\pi}(s'))$$

Figure 3: The Bellman Equation Update Rule

For each state in the environment, the utility is updated through this equation by summing over all possible actions from that state the expected future reward times the probability of taking that action in the environment. To be able to weight some state's reward more than others, we introduce a discount factor or gamma on the utility of the future state.

Frozen Lake

For the value iteration experiments I ran the algorithm for 300 rounds, where each round I set different amount of max iterations for the algorithm to run. After I got the policy from value iteration, I ran the policy on the environment for 500 iterations and calculated the average reward gained across the 500 iterations. For the frozen lake environment, I also calculated the success rate in reaching the goal state across the 500 runs of the policy on the environment. I tested the algorithm with different values of gamma. The results for the frozen lake environment are shown below.

Gamma	Max Average Reward	Max Success Rate	Average Iterations to Converge
0.9	1.863	78%	12
0.5	1.825	76.8%	7
0.1	1.779	75%	3

Table 1. Frozen Lake (Goal = 2.5, Hazard = -0.5) Results after running Value Iteration

In this version of the Frozen Lake environment, the goal state had a reward of 2.5 while the hazard states had rewards of -0.5. It was interesting to see that the maximum success rate across all the rounds and the maximum average reward decreased with smaller values of gamma. This could be because with a smaller gamma value we are weighting the expected future reward as less important, which would mean the utility values of states near the goal would be smaller. This would give you a policy that is slightly suboptimal and would explain the decreasing reward and success rate in reaching the goal. It is important to note that using a smaller discount factor does decrease the average amount of iterations for Value Iteration to converge to the optimal utility values. However, this change does present a tradeoff in terms of reward and number of successes the agent has in reaching the goal state.

I wanted to see how changing the reward function of the Frozen Lake problem impacted the optimal policy found by Value Iteration. I changed the reward for the goal state to a reward of 10 and the reward for the hazard state to a reward of -1.0. All other states in the environment

have a reward of 0. I expected the max average reward and the success rate in reaching the goal to increase because the goal state has a very high reward. I ran the same experiments as on the original frozen lake environment with different discount factors for the Bellman Equation.

Gamma	Max Average Reward	Max Success Rate	Average Iterations to Converge
0.9	0	0%	17
0.5	0	0%	10
0.1	0	0%	4

Table 2. Frozen Lake(G = 10.0, H = -1.0) Results

The policies found by value iteration with the new reward function performed very badly as they failed to reach the goal on all attempts. Changing the discount factor did improve the average number of iterations it took to converge to the optimal value table but it still resulted in a policy that could not reach the goal state of the environment. I was wondering what optimal policy the value iteration algorithm was finding. I displayed the environment map below along with the optimal action to take from each state according to value iteration.

S/U	F/U	F/U	F/U
F/L	**H/L**	F/L	**H/L**
F/U	F/D	F/L	**H/L**
H/L	F/R	F/D	**G/L**

Figure 3. Frozen Lake Map with State and Optimal Action (State/Action)
U- Up, D- Down, L- Left, R- Right

As shown in the table above, the agent's optimal policy is to always try and go up in the first row, which would just keep the agent in that row of the environment and thus away from the multiple hazard states in the second row of the map. It is also interesting to note that the state to the left of the goal has an optimal action of going down instead of going left into the goal state. This policy is very conservative and does more to avoid the negative reward in the hazard states than to advance the agent to the very high reward in the goal state. Here we see some of the drawbacks with value iteration. While the algorithm does find the optimal utility for each state which results in policies that minimize negative reward, it does not help find polices that advance the agent towards the goal state. The policy found may minimize the possible negative reward for the agent but may not necessarily get the largest reward possible, thus getting stuck at a local optimum.

Taxi

I used Value Iteration on the Taxi MDP. I was interested in seeing how long it took for the algorithm to converge on this environment since it has five-hundred states and a more

complex reward function. I used the same experiment procedure as for the Frozen Lake environment. For this environment, I focused more on the average reward found, the average iterations it took to converge, and the average number of steps taken by the agent when running the policy.

Gamma	Max Average Reward	Average Iterations to Converge	Average Steps Taken
0.9	-2.0	100	398
0.5	-2.0	20	398
0.1	-2.0	6	398

Table 3. Value Iteration Results on Taxi MDP

As seen in the table above, Value Iteration took much longer to converge with a discount factor of 0.9 when compared to both versions of the Frozen Lake Problem. We also see that the policies created by value iteration could not achieve an average reward higher than -2.0. I was not sure why these policies could not achieve a positive reward value. My initial reasoning was that since a negative reward is given for each time step the agent takes in the environment, the agent would want a policy that reaches the goal state in the fewest number of steps. However, we see that the agent is taking a relatively large number of steps in the environment despite the negative reinforcement that is given at each state. It is possible that since every state is giving a negative reward, the agent may find more utility in making an illegal drop-off and taking the quick exit than languishing in the environment any longer to find the +20 reward that comes from a successful drop-off. Here we have situation where value iteration is getting stuck at a local optimum in terms of utility, which results in the agent taking a policy that is not fully maximizing its reward.

Policy Iteration

As we saw in our two Markov Decision Processes, Value Iteration can run into trouble in environments with very large state spaces and with reward functions that penalize the agent for every step it takes in the environment. I was interested in seeing if using Policy Iteration on these environments saw improvement in the agent's average reward and success in reaching the goal. Unlike Value Iteration, Policy Iteration is trying to optimize the policy itself instead of optimizing the utility of each state in the environment. It calculates the policy at each step using the update rule shown below.

$$\pi^k[s] = \text{argmax}_a \sum_{s'} T(s,a,s') \, (R(s,a,s') + \gamma V[s'])$$

Figure 4. Update rule for Policy Iteration

For both environments, I ran the algorithm for 300 rounds. After I got the optimal policy from the algorithm, I ran the agent using this policy for 500 iterations and calculated the average reward gained across the 500 iterations. For the frozen lake environment, I calculated the success rate using the same method from the Value Iteration experiments.

Frozen Lake

The results for the frozen lake experiment can be seen in the chart below. I ran policy iteration on the environment with different values of gamma

Gamma	Max Average Reward	Max Success Rate	Average Iterations to Converge
0.9	1.654	70.6%	16
0.5	1.637	70.4%	16
0.1	1.611	69.4%	16

Table 4. Policy Iteration Results on Frozen Lake (G = 2.5, H = -0.5)

Here we see that the average maximum reward and maximum success rate using policy iteration are less than the average reward and success rate achieved using value iteration. However, if we look at the average reward across the 300 rounds that both algorithms ran we see that policy iteration is more consistent than value iteration in terms of average reward and success rate. Value Iteration starts out having a higher average reward but it quickly drops to zero after more and more episodes of the algorithm is run. On the other hand, Policy Iteration stays relatively consistent throughout all the trials.

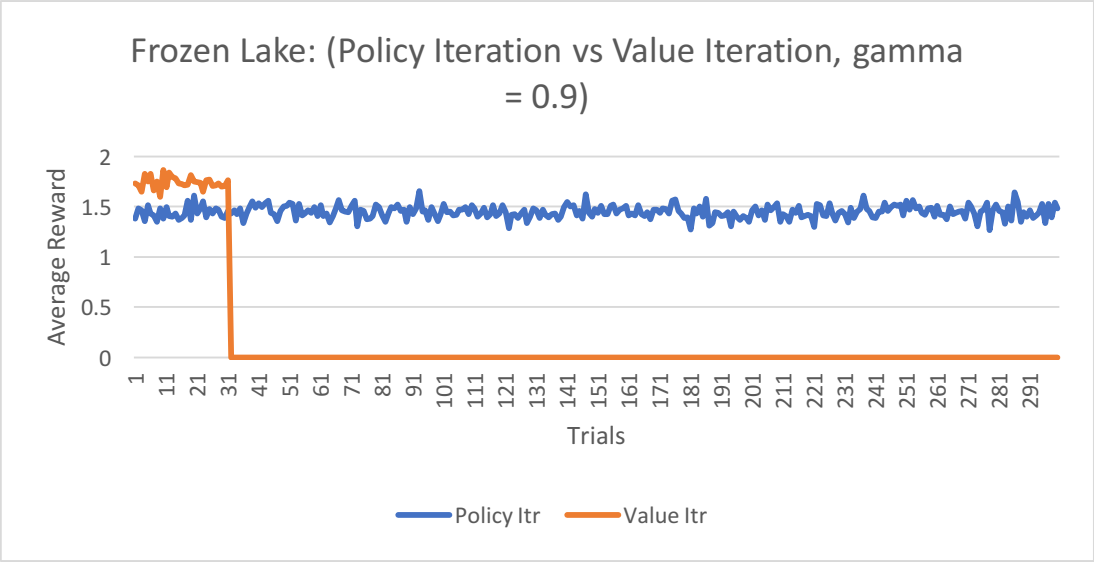


Figure 5. Comparison of Average Reward per Trial Between Policy and Value Iteration

Policy Iteration tends to be more consistent than value iteration because it waits for the policy itself to converge rather than the utility of each state to converge. It checks at every step whether the optimal action from a given state has changed and once it reaches a point where the action has not changed in any state, then the algorithm would have reached the optimal policy. On the other hand, the optimal value for each state may not necessarily give you the best policy possible. As noted before the utility values may converge to a local optimum for each state which would result in a policy that is less than optimal for the environment. We also see that Policy Iteration takes a consistent amount of iterations to converge to the optimal policy when

compared to Value Iteration. This is due to the nature of the algorithm as it does not need to optimize the utility values of every state like Value Iteration does.

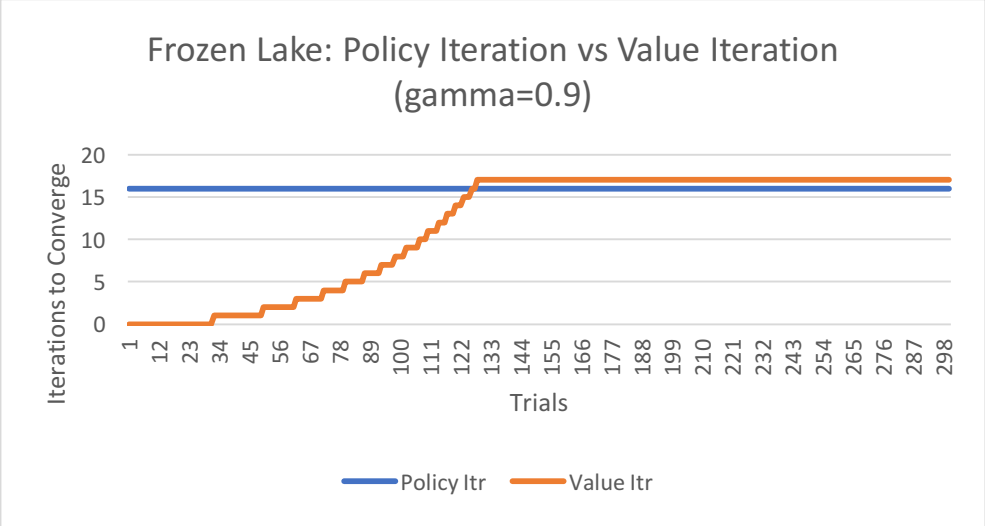


Figure 6. Comparison of Iterations to Converge between Policy and Value Iteration.

I also used Policy Iteration on this environment with the new reward function I defined earlier. The results of these experiments are shown below.

Gamma	Max Average Reward	Max Success Rate	Average Iterations to Converge
0.9	6.826	70.2%	16
0.5	6.652	69.2%	16
0.1	6.798	70.4%	16

Table 5. Policy Iteration on Frozen Lake(G = 10.0, H = -1.0)

Here we can see a major advantage in using Policy Iteration over Value Iteration for this environment. We saw that the policy created using Value Iteration was very conservative and had the agent avoid going down towards the goal state. Policy Iteration allowed the agent to reach the goal state despite the high probability of falling into the hazard zones of the environment. Since the algorithm is inherently trying to optimize the policy, it can find the best policy that safely navigates past the hazard spots in the environment while at the same time pushing the agent towards the goal state. We also see that the performance is consistent in terms of average reward and success across the three-hundred trials.

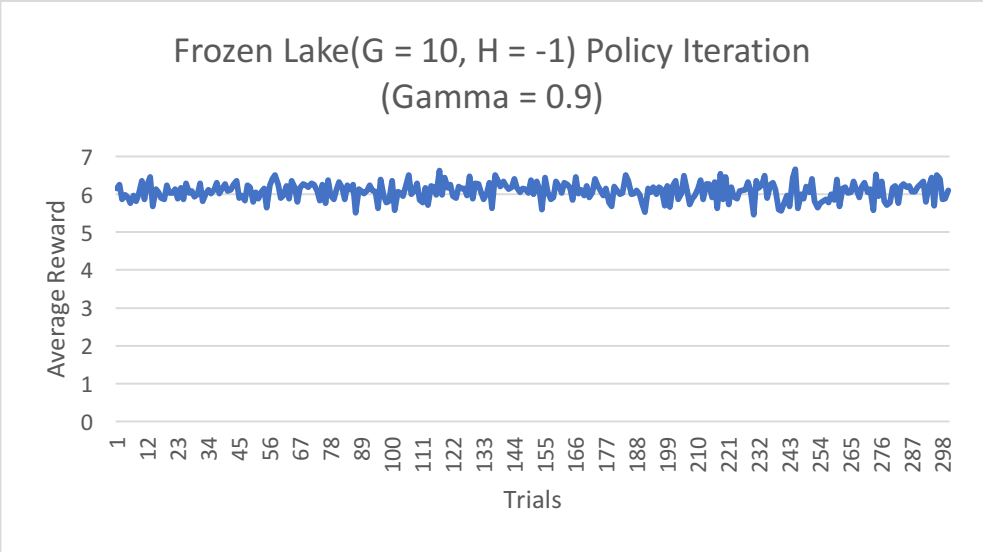


Figure 7. Average Reward Across 300 Trials

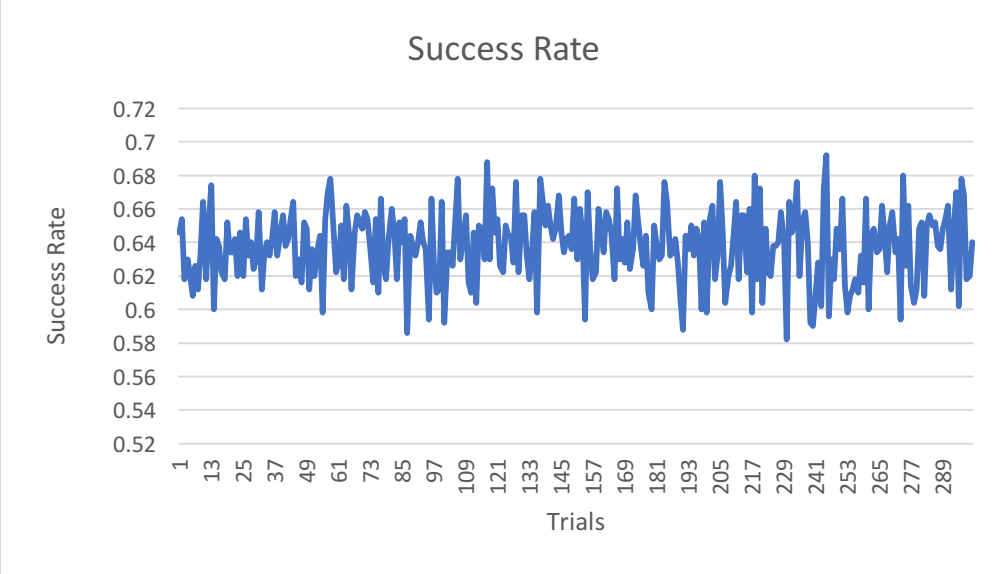


Figure 8. Success Rate Across 300 Trials.

Taxi

I ran Policy Iteration on the environment with the same method used on the Frozen Lake environment. The results for these experiments are shown below.

Gamma	Max Average Reward	Average Iterations to Converge	Average Steps Taken
0.9	-0.152	760	396
0.5	-0.656	747	396
0.1	-0.57	758	396

Table 6. Policy Iteration Results on Taxi Environment

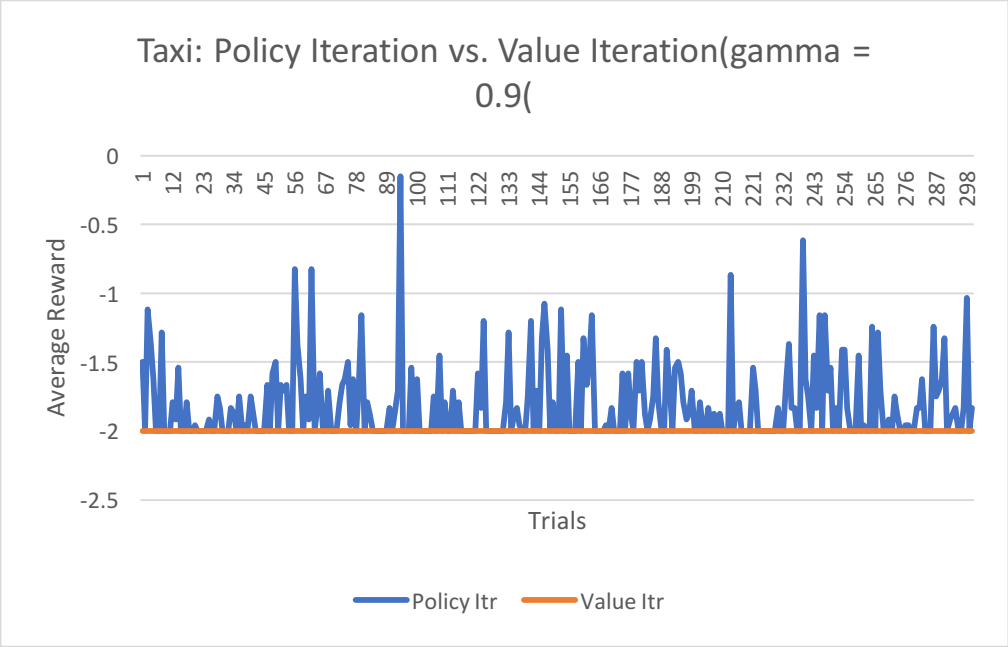


Figure 9. Comparison of Average Reward Between Value Iteration and Policy Iteration

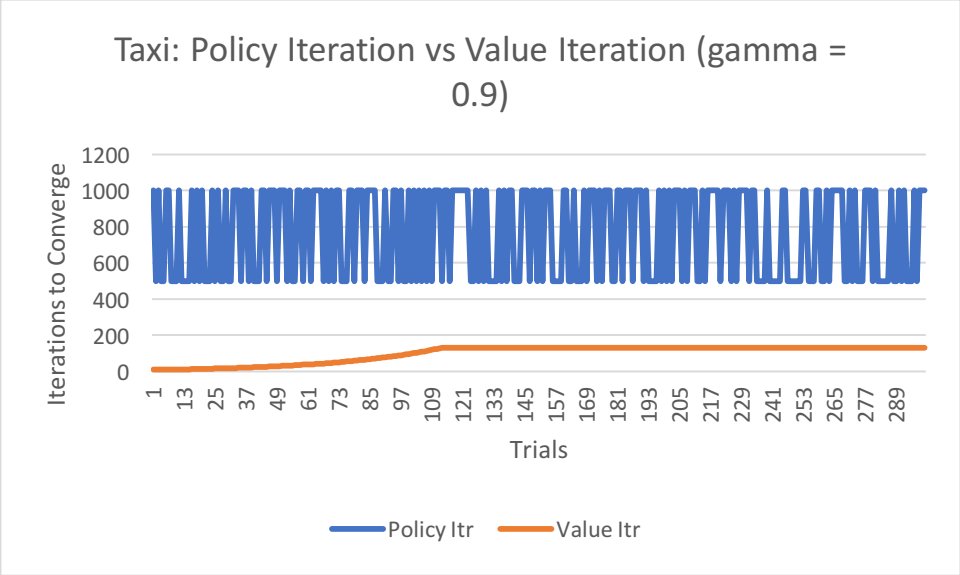


Figure 10. Comparison of Iterations to Converge Between Value Iteration and Policy Iteration

In terms of average reward, we see that Policy Iteration produces better results than Value Iteration as it can achieve an average reward above -2 and even gets reward as high as -0.4. It also resulted in the agent taking less steps in the environment towards the high reward state. However, Policy Iteration did take more iterations to converge than Value Iteration on this environment. I was unsure why this would occur given the inherent nature of Policy Iteration. I would have expected this algorithm to converge much faster than Value Iteration since an optimal policy can be found before the utility in each state is optimized. This result could be

related to the very large state space in the Taxi environment. Since the policy iteration algorithm is calculating new utility tables on every iteration, the state space size could result in very small changes the utility of every state from iteration to iteration. This would result in the algorithm taking many more iterations to converge than Value Iteration. However, the improved average reward from this algorithm is considerable enough to take the tradeoff between performance and run time that comes from using Policy Iteration on this environment.

Conclusion

In running Value Iteration and Policy Iteration on both MDP's it is clear to see some advantages and disadvantages in both algorithms. Value Iteration seems to work better on smaller environments with very simple reward functions but has inconsistent behavior in terms of runtime and average reward gained. Policy Iteration had more consistent behavior in terms of the number of iterations it took to converge and the average reward it achieved across all trials of the running the algorithm. The results from running Policy Iteration on both versions of Frozen Lake and the Taxi MDP show that the algorithm can handle more complex reward functions and reward functions that give you negative reward at every non-terminal state in the environment.