

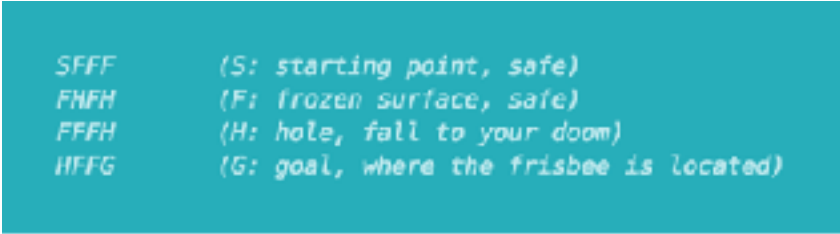
MDPs and Reinforcement Learning

Logan Allen

What MDPs did I choose and why are they interesting?

I used two MDPs created by OpenAI, a non-profit AI research company that is working to accelerate humanity's progress towards a safe form of artificial general intelligence. The two MDPs are a part of OpenAI's Gym, which is an open-source interface for reinforcement learning tasks. Gym was created so that researchers and independent developers could test and compare different approaches to a large variety of problems so that the state-of-the-art approach is always known. These MDPs are in large part interesting because they're a part of the OpenAI project. I decided to train on the FrozenLakev0 and FrozenLakev0-8x8 environments.

FrozenLakev0 is an MDP in which the agent controls the movement of a character in a grid world. Some tiles of the grid are walkable, and others lead to the agent falling into the water. Additionally, the movement direction of the agent is



SFFF	(S: starting point, safe)
FHFF	(F: frozen surface, safe)
FFFH	(H: hole, fall to your doom)
IFFG	(G: goal, where the frisbee is located)

uncertain and only partially depends on the chosen direction. The agent is rewarded for finding a walkable path to a goal tile with a reward of 1 but receives a reward of 0 if it falls into the water. The MDP is considered to be successfully solved if the agent gets an average reward of 0.78, as it is impossible to consistently solve the MDP with 100% accuracy. FrozenLakev0 is a very small MDP that consists of a 4x4 grid of squares, for a total of 16 states. It is an interesting problem because it is indeterminate and is used frequently as an example by people learning about reinforcement learning.

FrozenLake8x8-v0 is a much larger version of the FrozenLakev0 MDP. The MDP is considered to be successfully solved if the agent gets an average reward of 0.99. All of the dynamics are the same, but the environment is much larger. The MDP consists of a 8x8 grid of squares, for a total of 64 states. Despite the dynamics of the environments being the same, the size of FrozenLake8x8-v0 caused the algorithms to behave quite a bit differently on it.

Value Iteration - FrozenLakev0

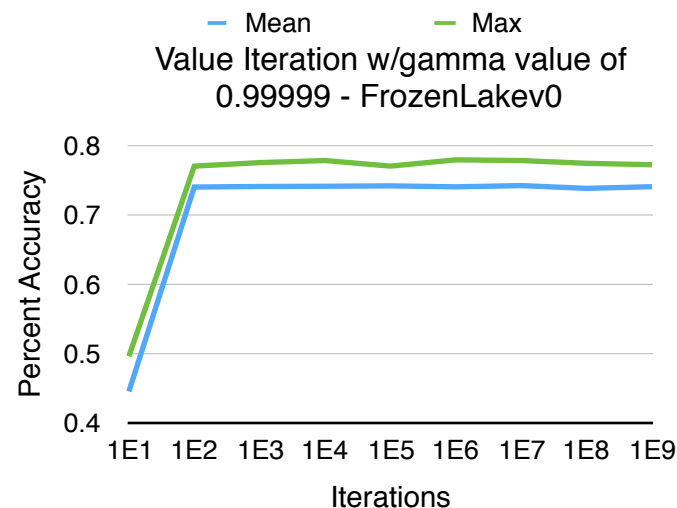
I modified the value iteration code that I found. I experimented with training value iteration policies with a number of iterations ranging from each order of magnitude $1E1$ - $1E8$. For each number of iterations, I created 100 different policies; after training the policies, I tested each one against the FrozenLakev0 MDP 1000 times so as to calculate the average reward value that each specific policy received. I refer to average reward as accuracy. I repeated this experiment methodology for gamma values of 0.8, 0.9, and 0.99999. I outputted my results as CSV files, then used Numbers to find the mean and max accuracies across each number of iterations that I tested.

Gamma	Max Accuracy	Mean Accuracy
0.8 (1E8 iterations)	0.494	0.448
0.9 (1E2 iterations)	0.769	0.732
0.99999 (1E7 iterations)	0.778	0.742

With a gamma value of 0.8, the accuracy remained incredibly low regardless of how many iterations the algorithm went through. Value iteration converged to approximately a 45-50% accuracy within 10 iterations, however, which is a very fast convergence. Value iteration is a very inexpensive algorithm with an MDP this small, however, so the tradeoff of fast convergence for lack of accuracy was not worth it.

The gamma value of 0.9 performed significantly better, and converged to a high level of accuracy $\sim 0.77\%$ within 100 iterations. Given that the MDP is considered to be successfully solved at 0.78%, these hyperparameters are an excellent combination. It's surprising that value iteration converged this quickly, as in the other trials, value iteration took significantly more iterations to converge than policy iteration.

The gamma value of 0.99999 took $1E7$ iterations to hit a maximum value, and converged to the highest possible accuracy of 0.78%. Value iteration requires access to the number of states, the number of actions, and the transition model of the MDP. These pieces of data are often not available in the real world, when there can be an infinite number of states, a large number of actions, and the transition model can only be approximated. However, in cases when the transition

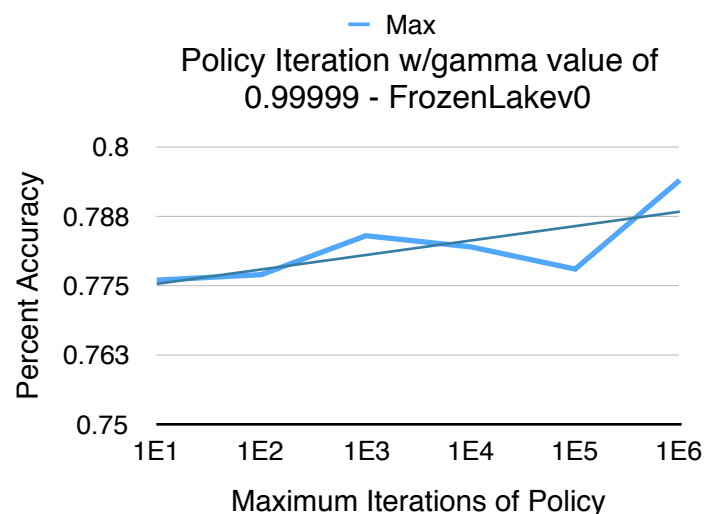
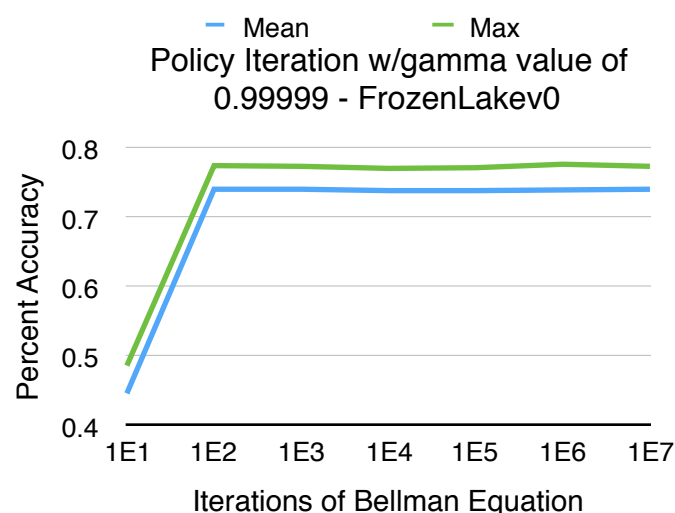


model is known, value iteration can be used to find the true value of being in every possible state. Once the true value of being in each state is known, a deterministic policy can be generated greedily that is optimal. Thus, optimal accuracy is guaranteed in the FrozenLakev0 environment where all of these parameters are known, but for the most part value iteration takes an incredibly long time to converge to the optimal policy.

Policy Iteration - FrozenLakev0

I modified the policy iteration code that I found. I experimented with multiple hyperparameters, such as the maximum number of policy iterations to perform, and the number of iterations of the Bellman equation to use for calculating the value function of the current policy. I experimented with policy iterations ranging through each order of magnitude from $1E1$ to $1E6$, and iterations of the Bellman equation from $1E1$ to $1E7$. For each number of iterations, I created 100 different policies; after training the policies, I tested each one against the FrozenLakev0 MDP 1000 times so as to calculate the average reward value that each calculated policy received. I refer to average reward as accuracy. I used this experiment methodology for the gamma value of 0.99999. This method of testing with policy iteration took a very long time to run, so I was unable to try other gamma values. I outputted my results as CSV files, then used Numbers to find the mean and max accuracies across each number of iterations that I tested.

I found that when iterations of the Bellman equation were $1E2$ or higher, the mean performance of policy iteration increased significantly. Interestingly enough, increasing the number of policy iterations did not seem to increase the average policy accuracy, but was on first glance slightly correlated with an increase in the accuracy of the maximum performing policy. Upon further analysis by changing the code to print out the final number of policy iterations once it



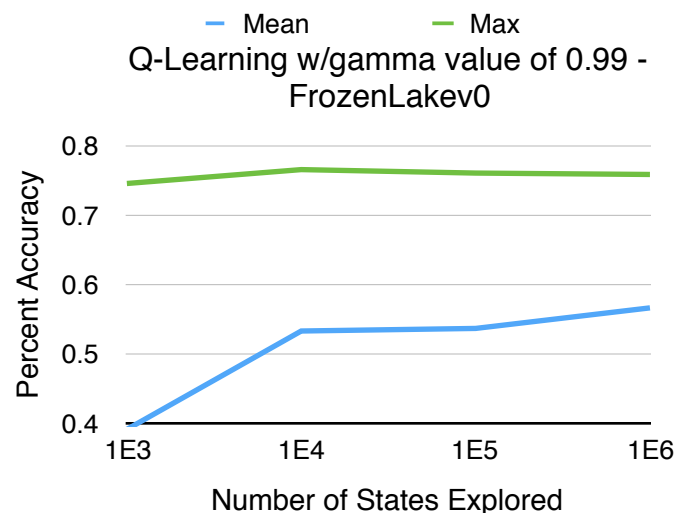
converged, I discovered that this correlation was not at all caused by the increase in the maximum number of allowed policy iterations, as policy iterations were converging after 1 to 3 iterations each time. I think the two largest factors in how policy iteration performed in this MDP were the number of iterations of the Bellman equation, as this is essentially just running value iteration on the policy, and how accurate the random starting policy was.

Policy iteration ran significantly faster than value iteration, converged faster, and received higher average levels of accuracy at lower numbers of iterations of the Bellman equation. After 100 iterations of the Bellman equation, policy iteration had an average accuracy of 0.775, whereas value iteration had an average accuracy of 0.742. Even though there were very few iterations of the policy, the performance was significantly higher and the function converged faster. Policy iteration also has access to the transition function, the set of states, and the set of actions, so it is able to perform equally well or better than value iteration in many cases.

Q-Learning FrozenLakev0

Q-Learning on average performed significantly worse than value iteration or policy iteration, primarily because both policy and value iteration assume that the agent has a large amount of domain knowledge. For policy and value iteration, they assume the agent knows all of the possible states it can be in and knows the exact probability of being able to transition from one state to another in what cases (completely understands all of the variables in the world). Q-Learning is more realistic, as it only assumes that the agent knows what the states of the world are and what actions the agent can take. Q-Learning directly estimates the value of each state then uses its estimates to form a policy.

In my experiments with Q-Learning, I ran each Q-Learning iteration 20-30 times, with a gamma value of 0.99. I varied the learning rate and the number of states explored. The number of states explored ranged from 1E1 to 1E6.



The table below shows the mean and max accuracies of Q-Learning performed 20-30 times per learning rate after exploring 100,000 states. The accuracies were found by using the policy found by Q-Learning to play 1,000 episodes of FrozenLakev0. The standard deviation was very high.

Learning Rate	Mean Accuracy	Max Accuracy
0.75	0.228	0.531
0.85	0.204	0.497
0.95	0.116	0.452
0.99	0.567	0.759

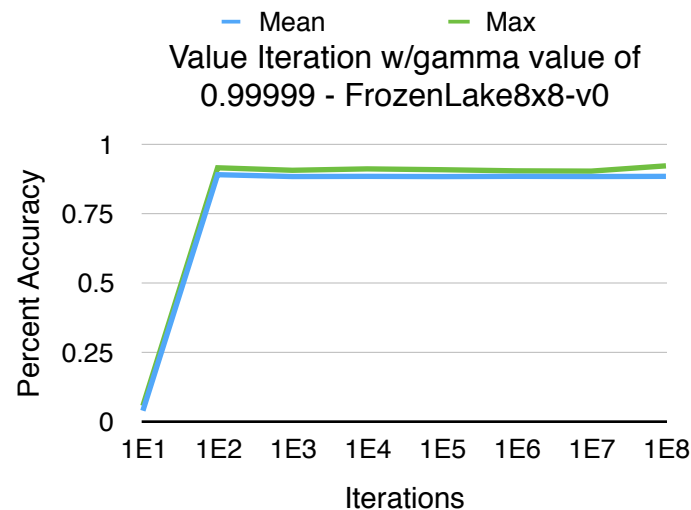
I performed more Q-Learning iterations with gamma values of 0.99 and a learning rate of 0.85, and varied the learning rate and the number of states explored. The number of states explored ranged from 1E1 to 1E6. However, the accuracy of Q-Learning had a very high standard deviation, which, depending on the number of iterations, ranged from 0.13-0.22, so after running Q-Learning 100 times where each agent only explored 1,000 states each, Q-Learning created a few agents performing almost as well as policy or value iteration. The max accuracy from this subgroup was 0.759.

Value Iteration - FrozenLake8x8-v0

I modified the value iteration code that I found. I experimented with training value iteration policies with a number of iterations ranging from each order of magnitude 1E1 - 1E8. For each number of iterations, I created 100 different policies; after training the policies, I tested each one against the FrozenLake8x8-v0 MDP 1,000 times so as to calculate the average reward value that each specific policy received. I refer to average reward as accuracy. I tried this experiment methodology with a gamma value of 0.99999. I outputted my results as CSV files, then used Numbers to find the mean and max accuracies across each number of iterations that I tested.

Iterations	Mean Accuracy	Max Accuracy
1E+01	0.042	0.060
1E+02	0.892	0.917
1E+03	0.886	0.908
1E+04	0.886	0.913
1E+05	0.885	0.910
1E+06	0.886	0.906
1E+07	0.886	0.905
1E+08	0.886	0.924

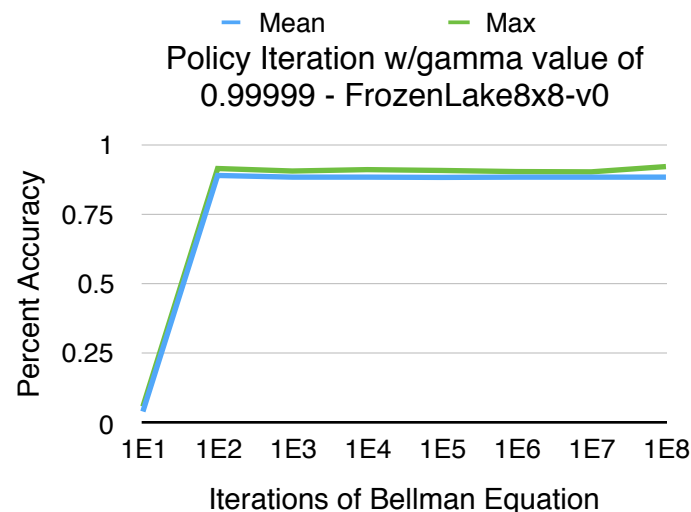
The gamma value of 0.99999 took 1E8 iterations to hit a maximum value, and converged to an accuracy of 0.924%, which is still 7 percentile points lower than the best possible value of 0.99%. Value iteration requires access to the number of states, the number of actions, and the transition model of the MDP. These pieces of data are often not available in the real world, when there can be an infinite number of states, a large number of actions, and the transition model can only be approximated. However, in cases when the transition model is known, value iteration can be used to find the true value of being in every possible state. Once the true value of being in each state is



known, a deterministic policy can be generated greedily that is optimal. Thus, optimal accuracy is guaranteed in the FrozenLake8x8-v0 environment where all of these parameters are known, but for the most part value iteration takes an incredibly long time to converge to the optimal policy. Because FrozenLake8x8-v0 is such a large environment, I did not run value iteration long enough to reach the optimal policy.

Policy Iteration - FrozenLake8x8-v0

I modified the policy iteration code that I found. I experimented with multiple hyperparameters, such as the maximum number of policy iterations to perform, and the number of iterations of the Bellman equation to use for calculating the value function of the current policy. I experimented with policy iterations ranging through each order of magnitude from 1E1 to 1E5, and iterations of the Bellman equation from 1E1 to 1E5. For each number of iterations, I created 100 different policies; after training the policies, I tested each one against the FrozenLake8x8-v0 MDP 1,000 times so as to calculate the average reward value that each calculated policy received. I refer to average reward as accuracy. I used this experiment methodology for the gamma value of 0.99999. This method of testing with policy iteration took a very long time to run, so I was unable to try other gamma values. I outputted my results as CSV files, then used Numbers to find the mean and max accuracies across each number of iterations that I tested.



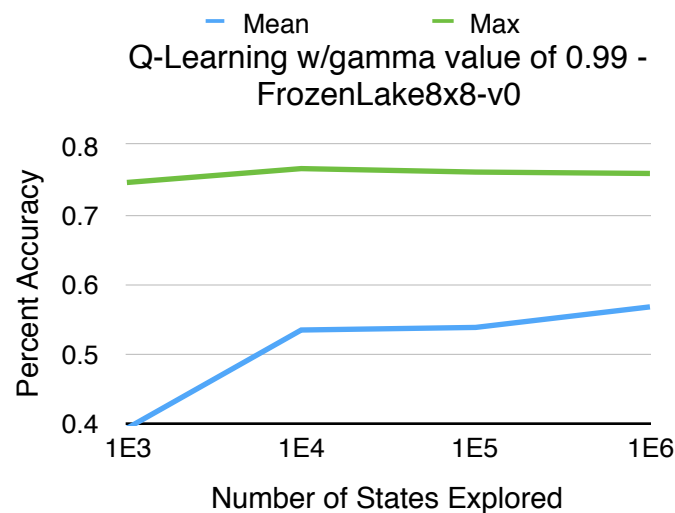
# of Bellman Iterations	Mean Accuracy	Max Accuracy
10	0.043	0.043
100	0.882	0.883
1,000	0.885	0.886
10,000	0.886	0.887
100,000	0.886	0.887

I found that when iterations of the Bellman equation were $1E2$ or higher, the mean performance of policy iteration increased significantly. Interestingly enough, increasing the number of policy iterations did not seem to increase the average policy accuracy, but was on first glance slightly correlated with an increase in the accuracy of the maximum performing policy. Upon further analysis by changing the code to print out the final number of policy iterations once it converged, I discovered that this correlation was not at all caused by the increase in the maximum number of allowed policy iterations, as policy iterations were converging after 2 to 6 iterations each time. I think the two largest factors in how policy iteration performed in this MDP were the number of iterations of the Bellman equation, as this is essentially just running value iteration on the policy, and how accurate the random starting policy was.

Policy iteration ran significantly faster than value iteration, converged faster, but received lower average levels of accuracy than value iteration, and significantly lower maximum levels of accuracy. After 100 iterations of the Bellman equation, policy iteration had an average accuracy of 0.882, whereas value iteration had an average accuracy of 0.892. The standard deviation of policy iteration performance was surprisingly low. Value iteration outperformed policy iteration repeatedly, with maximum values that were consistently higher than those of policy iteration. Policy iteration had a maximum accuracy of 0.887 at 100,000 iterations for policy iteration, whereas at the same number of iterations of the Bellman equation, value iteration had a maximum accuracy of 0.91.

Q-Learning FrozenLake8x8-v0

Q-Learning on average performed significantly worse than value iteration or policy iteration, primarily because both policy and value iteration assume that the agent has a large amount of domain knowledge. For policy and value iteration, they assume the agent knows all of the possible states it can be in and knows the exact probability of being able to transition from one state to another in what cases (completely understands all of the variables in the world). Q-Learning is more realistic, as it only assumes that the agent knows what the states of



the world are and what actions the agent can take. Q-Learning directly estimates the value of each state then uses its estimates to form a policy.

In my experiments with Q-Learning, I ran each Q-Learning iteration 20 times, with gamma values of 0.99 and allowed them to explore 100,000 states each.

The table below shows the mean and max accuracies of Q-Learning performed 20 times per learning rate after exploring 100,000 states.

Learning Rate	Mean Accuracy	Max Accuracy
0.75	0.119	0.324
0.85	0.441	0.862
0.95	0.178	0.402

I performed more Q-Learning iterations with gamma values of 0.99 and a learning rate of 0.85, and varied the learning rate and the number of states explored. The number of states explored ranged from 1E1 to 1E6. Q-Learning performed on average much worse than value or policy iteration, which makes sense as it has less information to work with. However, the accuracy of Q-Learning had a very high standard deviation, which was on average 0.282, so after running Q-Learning 50 times where each agent only explored 10,000 states each, Q-Learning created a few agents performing almost as well as policy or value iteration. The max accuracy from this subgroup was 0.858.