Markov Decision Processes

Sean Matthews
smatthews39@gatech.edu

Introduction

For this project, I worked with the reinforcement learning algorithm, Q-learning, as well as two other algorithms, value iteration and policy iteration. These were all implemented on two grid worlds. The first grid world was a cliff walking problem, and the second was an 18 row by 20 column grid, that represented a non-slippery lake, with holes that acted as broken ice. The grids were significant in size and layout so there would be a more visual and analytical comparison between the algorithms.

The Markov Decision Process (MDP) is a stochastic control process that is used in decision making in situations where the outcomes are partially random. If the process was not somewhat random, then the definite path to the reward, whether good or bad, would be the same every time the system ran. Giving a random factor in the problem ensured that the iteration and episodes were still random and chose different paths to attempt to converge. The MDP consists of a set of actions, a set of states, a probability that action ***a*** in state ***s*** will lead to some state ***s'***, an immediate reward from state ***s*** to ***s'*** [3]. Using the above, MDP can apply these policies to develop a set of expected rewards from the states or actions. It can continue to improve and iterate until it converges. To converge, the algorithm must not see any improvements for numerous iterations. Theoretically, policy iteration requires fewer iterations than value iteration. Policy iteration is normally faster if the transition probability is structured.

The implementation of the MDP problem was copied from the github repository linked in the references. It utilizes OpenAI Gym which is a toolkit for developing and comparing reinforcement learning algorithms [2]. Table 1 shows the parameters that were used in the grid world for the two algorithms as well as q-learner.

Table 1: Grid World Parameters

	Value Iteration	Policy Iteration	Q-learner
Max Iterations	1000	1000	500steps:2000Episodes
Discount Factor	[0: .9: .1]	[0: .9: .1]	[0: .9: .1]
Transition Probability	NA	NA	100% and 33%
theta	0.00001	.0001	0.0001
ϵ	NA	NA	[0.1, 0.3, 0.5]
α	NA	NA	[0.1, 0.3, 0.9]

The Q-initial values give a initial capability of how exploratory the algorithm needs to be. In example, if by the odd chance these initial values were extremely close to the actual value of a ran q-learner, it would be expected that the episode would converge in a minimal amount of steps, possibly even one step. These values are essentially estimates of the sum of future rewards. The stochasticity is shown as within the transition probability above. I have performed experiments with a deterministic q-learner as well as a probabilistic q-learner. I will discuss this in the Frozen Lake experiment.

The following experiments for policy and value iterations are considered converged if the delta across all seen states stays near constant by the defined theta value, for more than 10 steps.

Cliff Walking (4x12)

For the cliff walking problem, the grid world is on the smaller side with just 4 rows and 12 columns. The goal of this word is to get from one side of a cliff to the next without walking off the cliff. I chose to make this problem less optimal than fully deterministic and added wind to the problem. This adds a factor of probability into the decision of where you are stepping. In example, if I were right on the edge of the cliff, and a gust of wind occurred, it is not probable, but possible that I get blown off the cliff even though it was not the optimal action I could take. Figure 1 shows the grid world that was used in this exercise with the algorithms and q-learning.

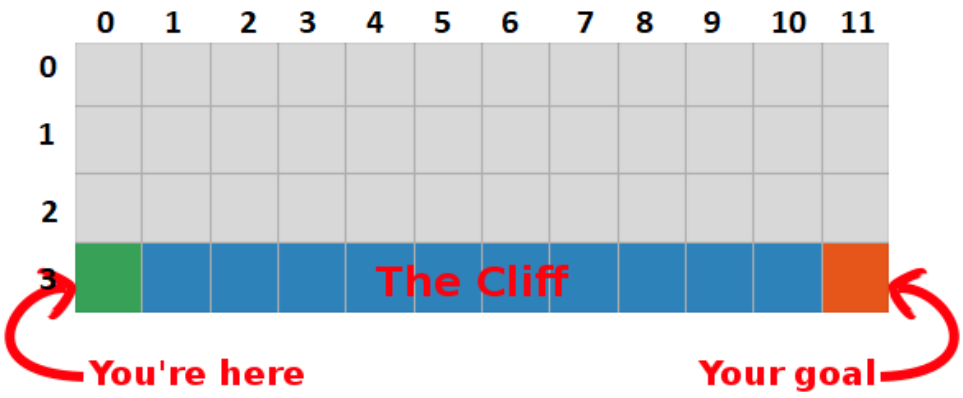

Figure 1: Cliff Walking Grid World [1]

Value Iteration

Value iteration uses the Bellman equation and calculates all the states until it converges. The neighboring states are calculated based on discounted awards and the algorithm iterates through the utility values until all of the states have been evaluated. I used a range of discount factors from 0 – 0.9. The difference between this simple variable was quite drastic. Table 2 shows the maximum and minimum reward for each of these discount factors after the value iteration had converged.

Table 2: Value Iteration Maximum and Minimum Values

γ	0.0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
Maximum	100.0	111.11	125.0	142.86	166.67	200.0	250.0	333.33	500.0	1000
Minimum	-1.0	-1.11	-1,25	-1.43	-1.66	-1.95	-1.95	1.33	29.7	275.25

The results in Table 2 are expected. The discount factor, **γ**, is a constant between 0 and 1 that determines the relative value of delayed versus immediate rewards. The closer **γ** is to 1, the future rewards are given greater emphasis relative to the immediate reward. As **γ** decreases, the impact of the future rewards relative to the immediate reward decreases exponentially as a factor of **γ**. With our algorithm set to include 1000 steps, it is no surprise that our max reward would be 1000, given a 1 reward per step with a high **γ**.

Below are the last iterations for discount factors 0.1, 0.5 and 0.9.

Figure 2: Last Value Iteration , γ=0.1, 0.5, 0.9

The amount of steps for convergence is low with a lower discount factor as the immediate reward is significantly more weighted. Giving the possible directions a more defined route to the end reward.

Policy Iteration

In Policy Iteration, the utility values are calculated by creating arbitrary policies. It attempts different policies and checks to see if the new policy can return a larger utility value. This iteration continues until there is not improvement per iteration in the utility value. It is a converging algorithm, but it requires more time and calculation by solving equations for each state.

Figure 3: Last Policy Iteration, γ=0.1, 0.5, 0.9

Each of the policy iteration episodes converged within 8 steps, and the γ showed just as effective in policy iteration as it did in value iteration.

The timing for both iteration algorithms are shown in Table 3.

Table 3: Value & Policy Iteration Timing

Value Iteration (Average)	Policy Iteration (Average)
.00083 seconds	.00235 seconds

From this timing it supports the hypothesis that the policy iteration takes significantly more time than the value iteration. Policy iteration solves system of equations, and loses efficiency with the increasing calculations demanded by the policies.

Q-Learning

Q-learning doesn't use domain knowledge to calculate the transition function. Initial Q values are assigned to two different values, random and 0. The learning rate for my experiments are 0.1, 0.5 and 0.9. The randomness of my learner, ϵ, is set to use 0.1, 0.3, 0.5. Using Q-learning, the agent will take the actions to the states and this will assign new q values based on immediate and delayed rewards [4]. Q-learning for the cliff walking grid world performed quite well. The wind probability applied to this problem is 0.1. The wind affects how the probability of determining the next step will be calculated.

The timing for Q-Learning took a very long time, relatively speaking, the first episode. The first episode, as you can see in Figure 4, took approximately .030 seconds, while throughout the rest of the episodes, no single episode took any longer than .005 seconds.

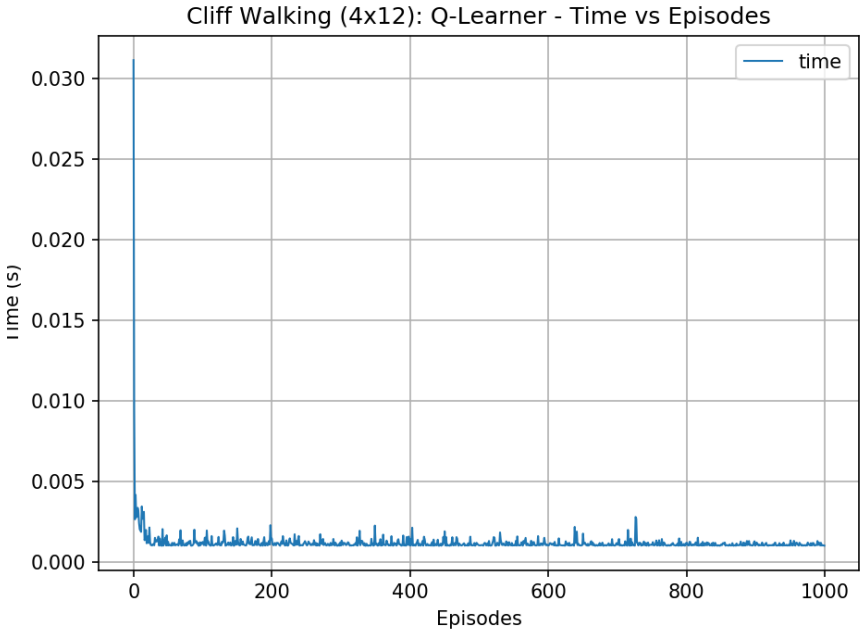

Figure 4: Timing for Q-Learner Episodes

The learner most likely took additional steps in the first episode as it looked at the grid world. Figure 5 shows the reward for the 1000 episodes. The trend shows that the learner had some difficulty due to some of the transition probability, but ultimately converges to a reward just below 6. The Q-Learner shows promising results as the reward tends to move to only to rewarding values large than 4 and eventually 5.

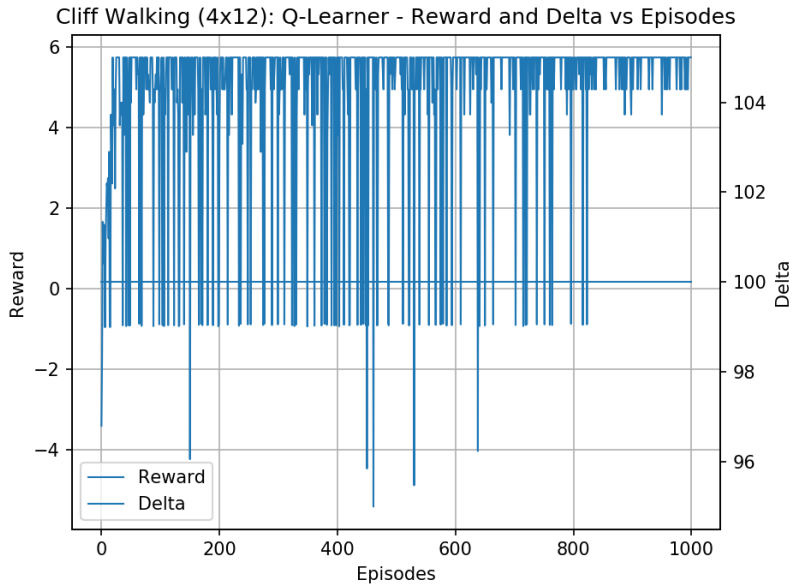

Figure 5: Reward and Delta vs Episodes

The above experiment utilized the best parameters. Alpha was 0.9, Q-Inits was 0, Epsilon was 0.1, decay of 0.0001 and a discount factor of 0.3.

Frozen Lake Large (18x20)

This graph consisted of a large grid, 18 rows by 20 columns, and can be seen in Figure 6. The green box indicates the start location and the yellow box represents the goal. For the 18x20 large grid world, the policy and value iteration converged within 8 steps and 53 steps respectively. I modified the world from the original code that I implemented to include more gaps and less rows to decrease computation time, and q-learner compilations.

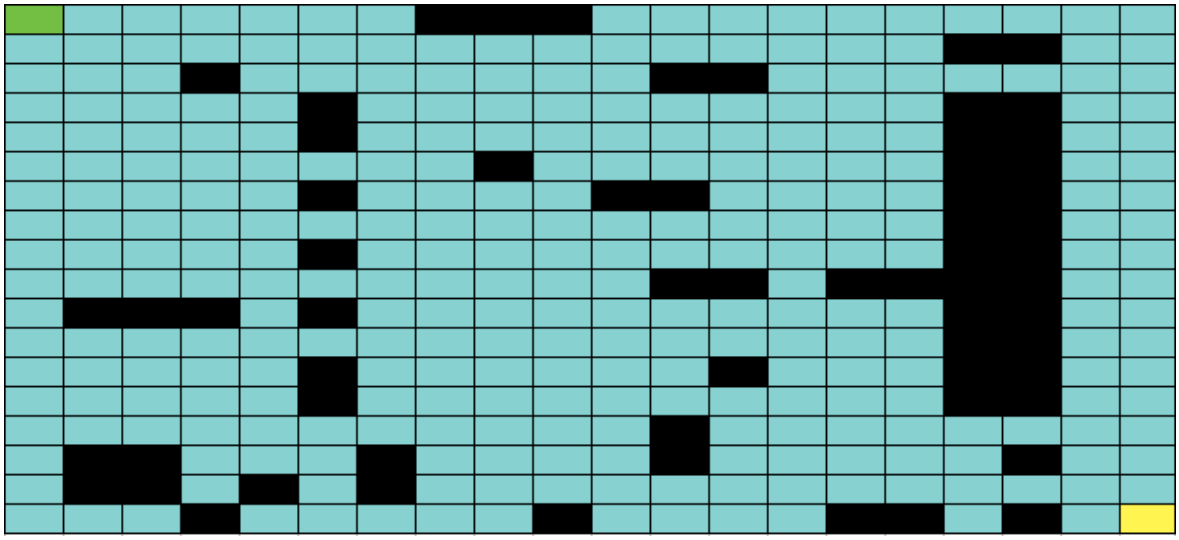

Figure 6: Large Frozen Lake

Value Iteration & Policy Iterations

As I had before for the small problem, I used a variable γ from 0.0 – 0.9. Below are the values that were accumulated as the reward.

Table 4: Value Iteration Maximum and Minimum Values

γ	0.0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
Maximum	0.33	0.35	0.36	0.38	0.4	0.43	0.47	0.52	0.59	0.71
Minimum	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

Again as the value iterator in the small grid world, as γ increased the overall reward also. The value iterator reached its convergence within a maximum 53 iterations. That is quite quick for such a large grid world. In comparison, the policy iterator surprisingly converged within 7 iterations. In terms of timing, the value iterator took about 5 seconds for each episode. while the policy iteration

increased from the first episode (0.67 seconds) to the last episode (5.59 seconds). The last episode for the policy experiments took just about nearly as much time as the value iterator.

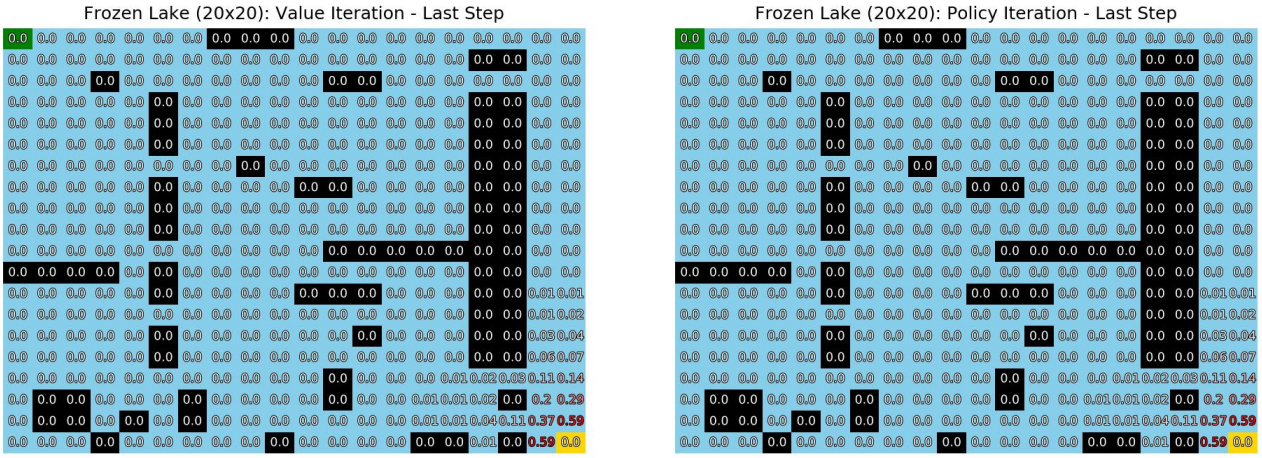

Figure 7: Value and Policy Iteration

What is most surprising about the images in Figure 7, is that they have all the same rewards. They converged on different steps but have the same reward values.

Q-Learning

The Q-learner analysis will include comparisons of the following:

- Q-Inits values
- Learning Rate
- Transition Probability

Other analysis were completed with the Q-Learner but were excluded from this report. Some more information about this experiment is that for each step that the learner steps onto a 'safe' action, the learner is awarded a -0.1. If the learner completes the world, it is awards 1.00.

First off, the initial q values were given randomly. Figure 8 is the result of assigning random values. The other run used 0 as an initial value for each state. For consistency for this experiment, the figures will be using the parameters in Table 5.

Table 5: Q-Inits Analysis Parameters

α	ϵ	ϵ Decay	Discount Factor
0.5	0.5	0.0001	0.8

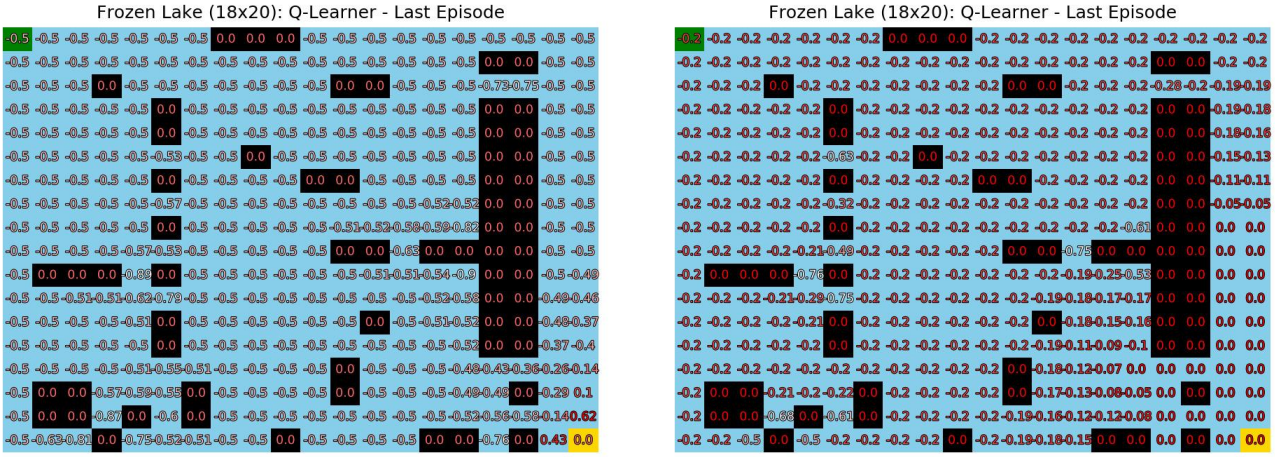

Figure 8: Random Q-Inits (Left), 0 Q-Inits (Right)

When initial values of Q are arbitrarily initialized, the agent actually received the goal reward for numerous episodes. However, when the values are initialized to 0, the agent never reaches the reward within the episodes. This test showed to not converge to the reward, but it did converge to a less than optimal reward.

The learning rate looked to make a large difference in the return for the large frozen lake. Figure 9 shows the difference that learning rate plays on the learning. Table 6 are the parameters used for the other variables

Table 6: Q-Inits Analysis Parameters

Q-Inits	ϵ	ϵ Decay	Discount Factor
0.0	0.5	0.0001	0.8

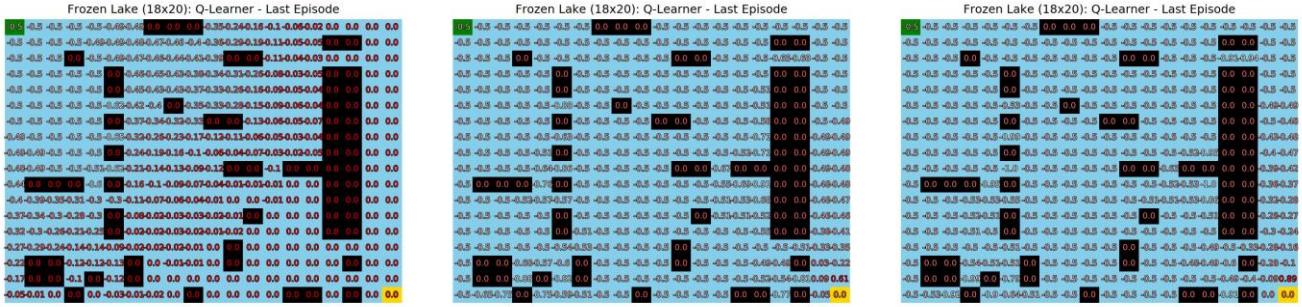

Figure 9: α: 0.1 (Left), α: 0.5 (Center), α: 0.9 (Right)

When the learning rate is 0.1, the converged reward is suboptimal. It converges to a hole as it never reaches the goal. When α is 0.5, the learner does reach the reward, but concludes with a 0.61

reward in the end boxes. An α of 0.9 showed to be very effective and the reward converged to 0.89 reward. When variablizing α, it shows to be one of the larger contributors to a successful learner.

The last experiment I ran was transition probability. The first experiment was ran with a 100% probability, and the second was ran with a 33% probability. What this means is that although the direction has a higher value, it still has a 33% chance of going that direction, making it rather random. Figure 10 has the two experiments, and the left side shows the deterministic test, and the other shows the lower probability.

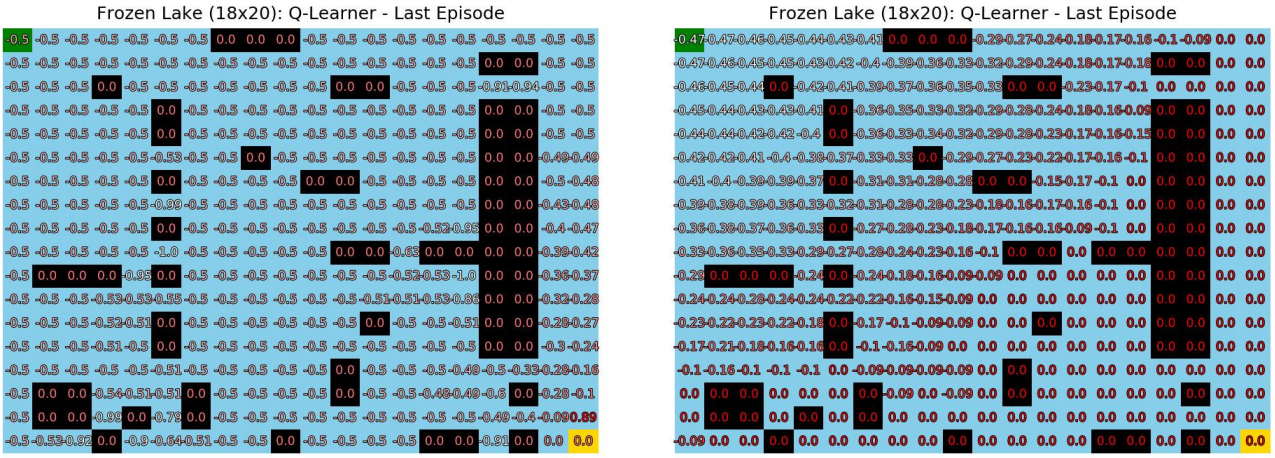

Figure 10: Transition Probability, 100% - Left, 33% - Right

Before completing this experiment, I hypothesized that the deterministic model would significantly outperform the lower probability model. Above the deterministic model reaches the end goal reward and succeeds with a 0.89 reward. The other experiment showed that it never reached and converged to the goal.

After these problems, one thing I noticed was that as the discount factor increases, the number of steps to convergence also increases. This increases computation and the run time.

References

1. Vazquez, Lucas. "Understanding Q-Learning, the Cliff Walking Problem." *Medium*, NeuroAscent.ML, 7 Apr. 2018, medium.com/init27-labs/understanding-q-learning-the-cliff-walking-problem-80198921abbc.
2. OpenAI. "A Toolkit for Developing and Comparing Reinforcement Learning Algorithms." *Gym*, gym.openai.com/.
3. "Carnegie Mellon School of Computer Science." *Carnegie Mellon School of Computer Science*, www.cs.cmu.edu/.
4. Mitchell, Tom M. *Machine Learning*. McGraw Hill, 2017.

5. https://github.com/cmaron/CS-7641-assignments/tree/master/assignment4