Dhsieh6@gatech.edu

**Reinforcement Learning Assignment**

For this assignment, I chose two similar problems: the frozen lake and the taxi problem. Both problems involve a grid world, where each run ends when a destination is reached.

The Frozen Lake problem involves a lake broken up into a grid. The grid represents the state of the lake, which could either be safe, a hole, or the goal. You are only rewarded for reaching the goal and nothing else. The world itself is stochastic because the ice is slippery, and you don’t always go in the direction you want. A given run ends when you either reach the goal or fall into a hole. In this case, there are 16 states, since the lake is split up into a 4x4 square.[[1]](#footnote-1) There are four actions: up, down, left and right.

The taxi problem, on the other hand, involves trying to successfully pick up and drop off a passenger, with each run representing one successful trip. This is a larger problem space, with 500 distinct states. The taxi can be in any of 25 positions, and there are also states that describe passengers as well as their destinations. There are 6 actions: up, down, left, right, pickup, and drop-off. Because we are trying to train the learner to pick up passengers and get them to their destination as quickly as possible, we reward -1 for each action and an additional +20 for delivering the passenger. We don’t want the taxi to pick up strangers, so we reward it -10 for picking up or dropping of randomly.[[2]](#footnote-2)

The two problems, although seemingly similar in how they break up the world into a grid, represent different challenges when both defining the state and action pairs, as well in configuring an appropriate learning algorithm. The frozen lake is stochastic since the ice is slippery, whereas the taxi problem is deterministic since the taxi driver goes where he/she wants at each decision point. However, the frozen lake is simpler in that there are fewer states to iterate through, whereas the taxi problem has more states since it has to deal with the added complexity of picking up and dropping off passengers, as well as doing so as quickly as possible. We will explore in the rest of this assignment how these different characteristics impact the learning algorithms.

**Implementing Value Iteration, Policy Iteration, and Q-Learning**

To start, I leveraged an existing code tutorial[[3]](#footnote-3) to fashion my value iteration and policy iteration algorithms. Value iteration involves iterating through every state and choosing the action that results in the maximum value. The value is a combination of the past known value and a discounted newly learned value. By using this value calculation, we are able to propagate learnings made in the future to earlier states. Policy iteration, on the other hand, takes a different approach in that it starts with a policy and iterates through each state within that policy, updating the policy to maximize the action taken. This results in fewer iterations, but more computationally expensive iterations. Finally, Q-Learning is a form of reinforcement learning that does not require knowing a model that describes the environment, and instead learns what to do by playing through the environment multiple times. It typically requires more iterations combined with an exploratory strategy in order to hit relevant actions. For Q-Learning, I leveraged an existing Q-Learning algorithm I had built for a prior Georgia Tech class and adjusted it to work with both the Frozen Lake and Taxi problems.

*Value Iteration applied to the Frozen Lake*

Value iteration takes two parameters: theta and a discount factor. Theta is the cliff that determines when the model should stop iterating. Discount factor determines how quickly the model should learn based on new information. To start, I wanted to understand how different discount factors might impact how quickly the model could learn.

|  |  |  |  |
| --- | --- | --- | --- |
| **Theta = 0.0001** | **Discount = 0.9** | **Discount = 0.99** | **Discount = 0.999** |
| # of iterations required | 35 | 131 | 178 |
| Processing time | 0.01s | 0.04s | 0.06 |

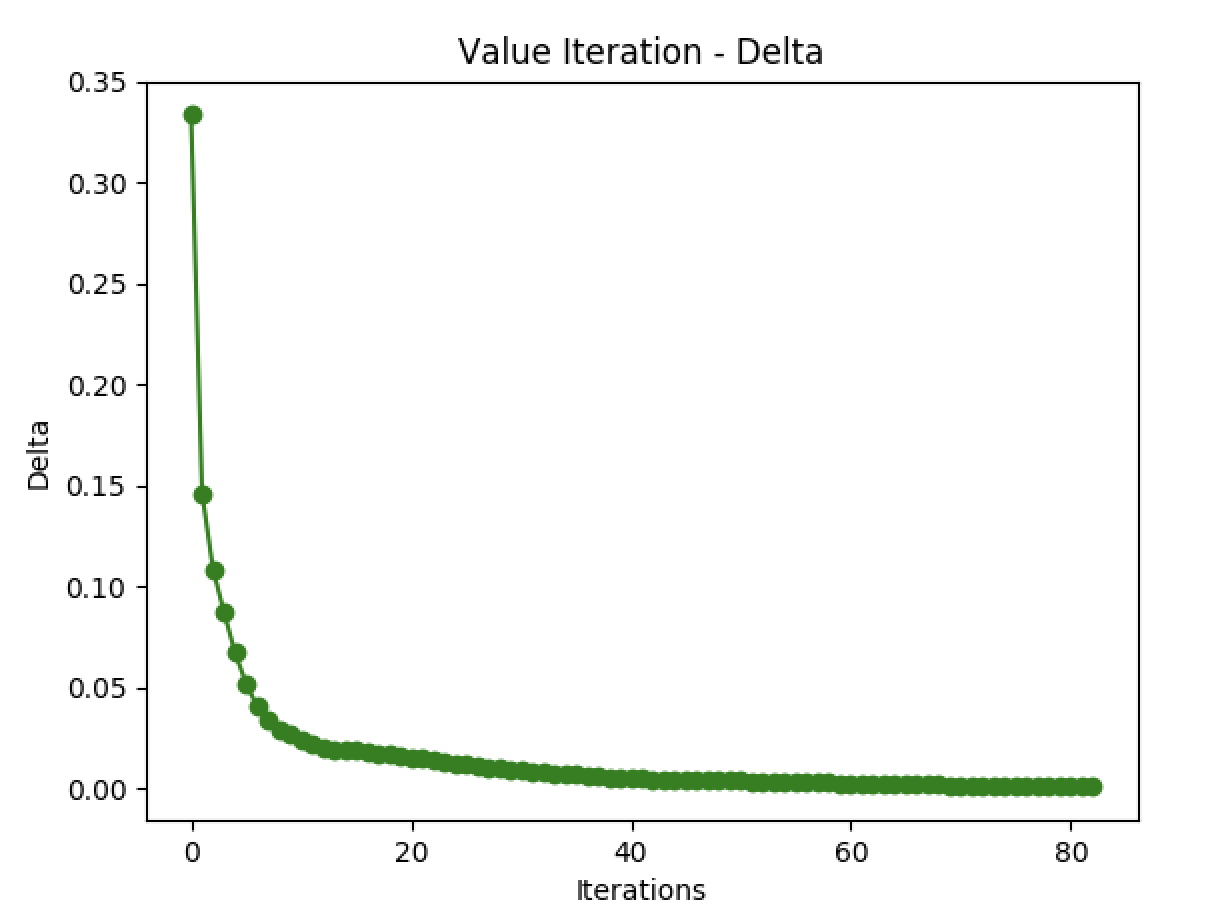
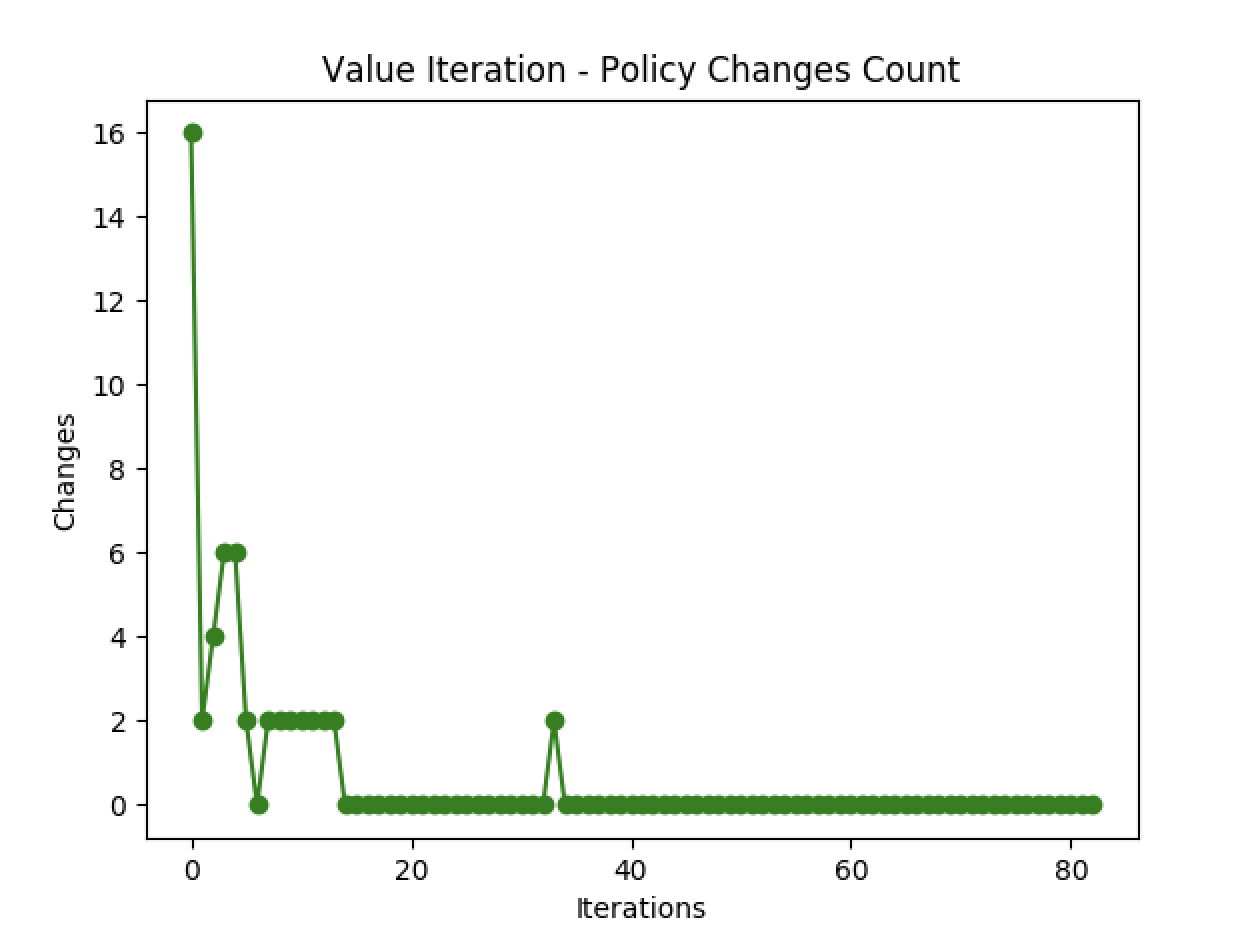
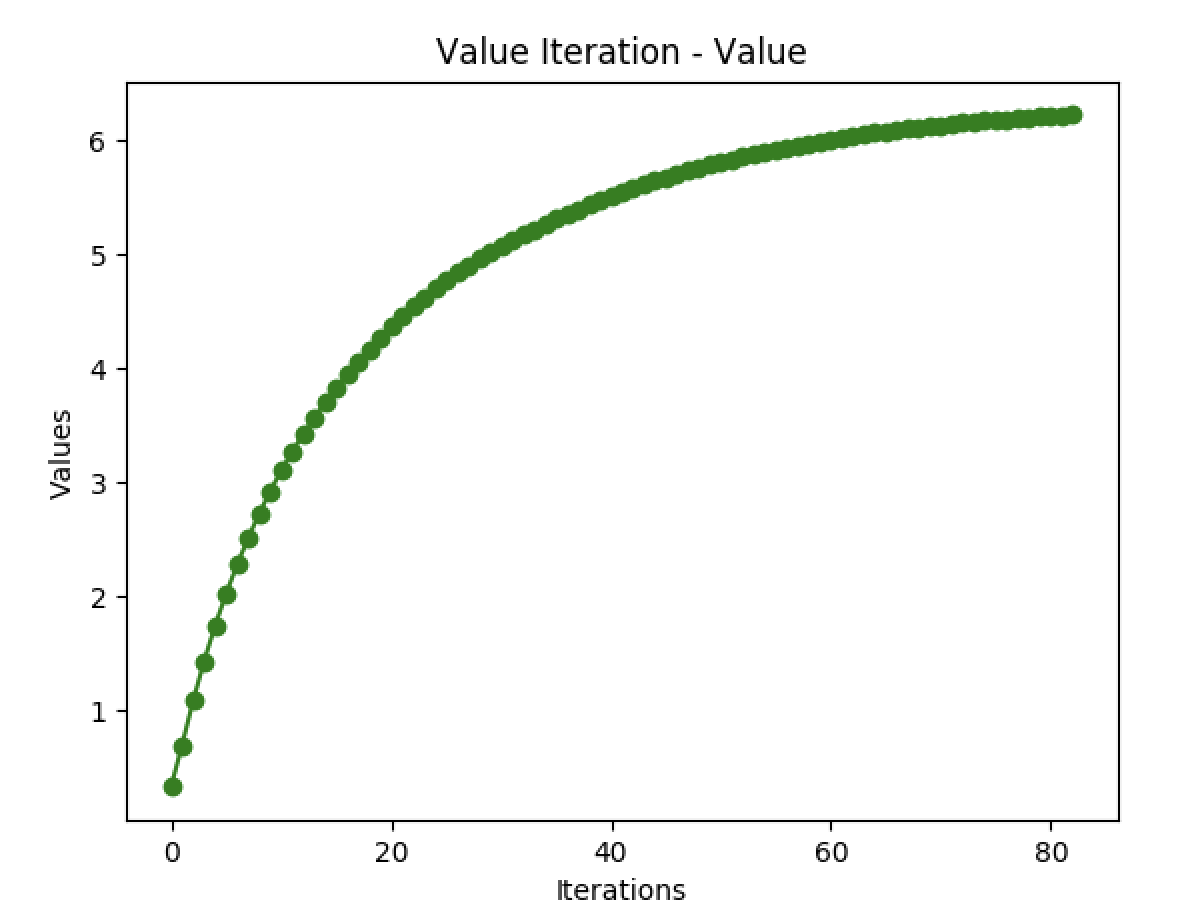
As expected, lowering the discount factor means that the model converges quicker. Processing time is lower and the # of iterations required to converge delta to theta is significantly reduced. However, do we trade off the most optimal policy by learning too quickly? To test this, I ran all three discount factors and compared the policy that it converged to. Indeed, the policy discovered by discount factor 0.9 has two states where it chooses a different action from the policy discovered by discount factor 0.99 and 0.999. The policies discovered by discount factor 0.99 and 0.999 are the same. This indicates that a discount factor of 0.9 is too large and doesn’t give the algorithm adequate time to find the optimal path.

Similarly, I wanted to see if adjusting theta had similar characteristics. Changing theta is similar to changing discount factor, since a larger theta reduces the number of iterations the algorithm runs. In the following table, I ran theta for three different values, keeping discount factor at 0.99 since based on the prior analysis, that discount factor allowed for a good trade-off between accuracy and processing time. The results were similar to changing discount factor, with the largest theta converging too quickly, resulting in two actions that were different from the policies discovered by value iterations with lower thetas.

|  |  |  |  |
| --- | --- | --- | --- |
| **Discount = 0.99** | **Theta = 0.01** | **Theta = 0.001** | **Theta = 0.0001** |
| # of iterations required | 29 | 83 | 131 |
| Processing time | 0.02s | 0.05s | 0.08 |

By combining the two analyses above, I decided to use a discount factor of 0.99 and theta of 0.001 to strike a balance between processing speed and accuracy.

Below are the results achieved using value iteration with a discount factor of 0.99 and theta of 0.001. As you can see, value iteration converges to an optimal policy with decreasing delta and increasing value. The delta graph shows how the change in value changes over each iteration. The policy changes count graph counts how many actions were different across policies with each iteration. The value iteration graph shows how the value changes over each iteration. You can see that in the policy changes graph, there is an anomaly between 30-40 iterations where the policy changes slightly before converging. This is likely because the frozen lake is stochastic, so at the 30-40 iteration phase, the discount is still large enough for random changes to impact the overall policy. That is why we have to iterate more times to reduce the impact stochasticity has on the final policy result.

*Policy Iteration applied to the Frozen Lake*

Unlike value iteration which iterates across states and actions, policy iteration iterates across policies to identify the best policy. Policy iteration takes just one variable – a discount factor. I ran a similar analysis to what I did above, basically trying different levels of parameters to identify which parameter allows the best trade-off between accuracy and processing time.

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Discount = 0.9** | **Discount = 0.99** | **Discount = 0.999** |
| # of iterations required | 2 | 3 | 3 |
| Processing time | 0.02s | 0.09s | 0.11s |

As with value iteration, a discount factor that is too high makes too much of a trade-off between processing speed and accuracy. When comparing the policies, the policy found by discount factor 0.9 chooses two different actions compared to the policies found by 0.99 and 0.999, whereas the policies found by 0.99 and 0.999 are exactly the same. In this case, policy iteration needs less iterations since it is greedily identifying the most effective actions for each policy. However, each iteration is computationally more expensive. As a result, the processing time is actually longer than value iteration.

*Value Iteration compared to Policy Iteration in the Frozen Lake*

Ultimately, policy iteration took fewer iterations to converge, but took longer to process than value iteration because each iteration is computationally more expensive. Both value iteration and policy iteration resulted in the same optimal policy.

*Value Iteration applied to the Taxi Problem*

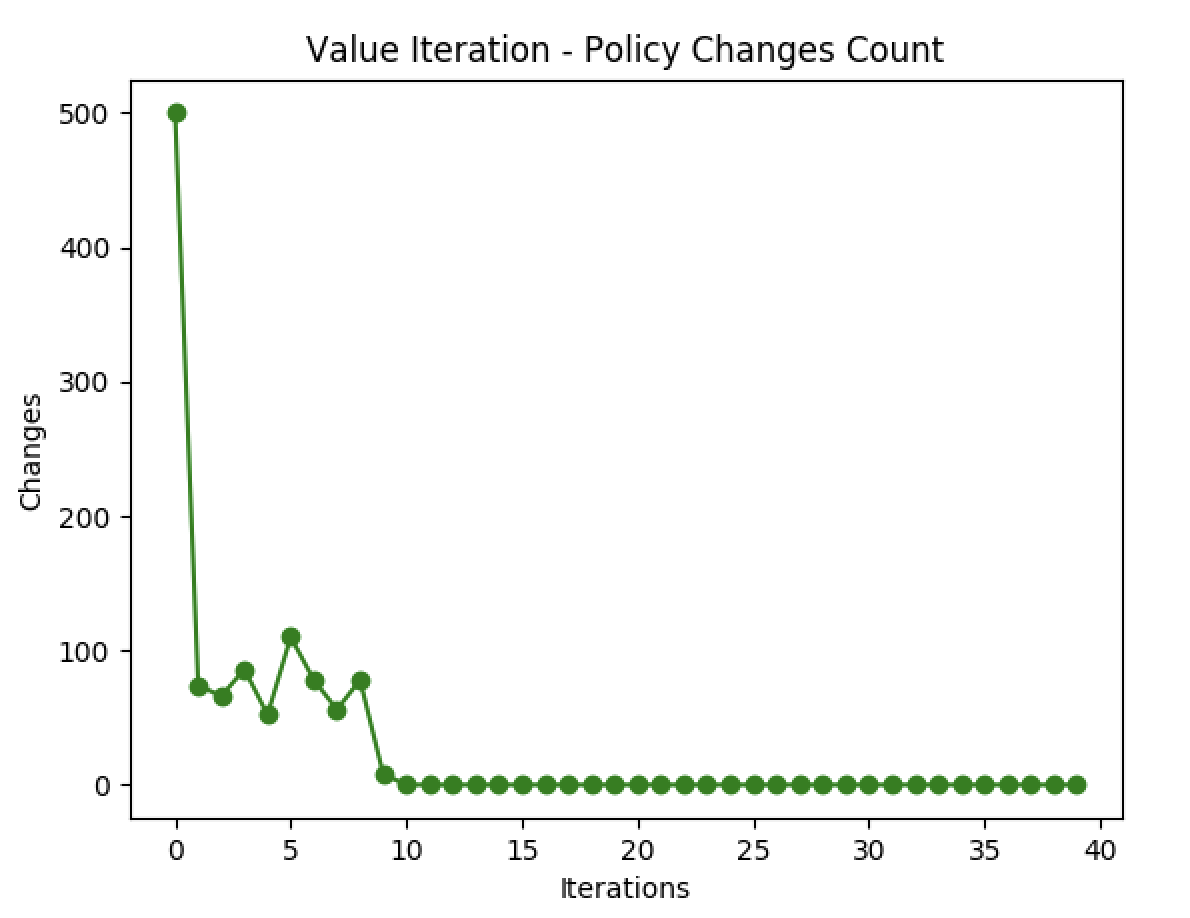
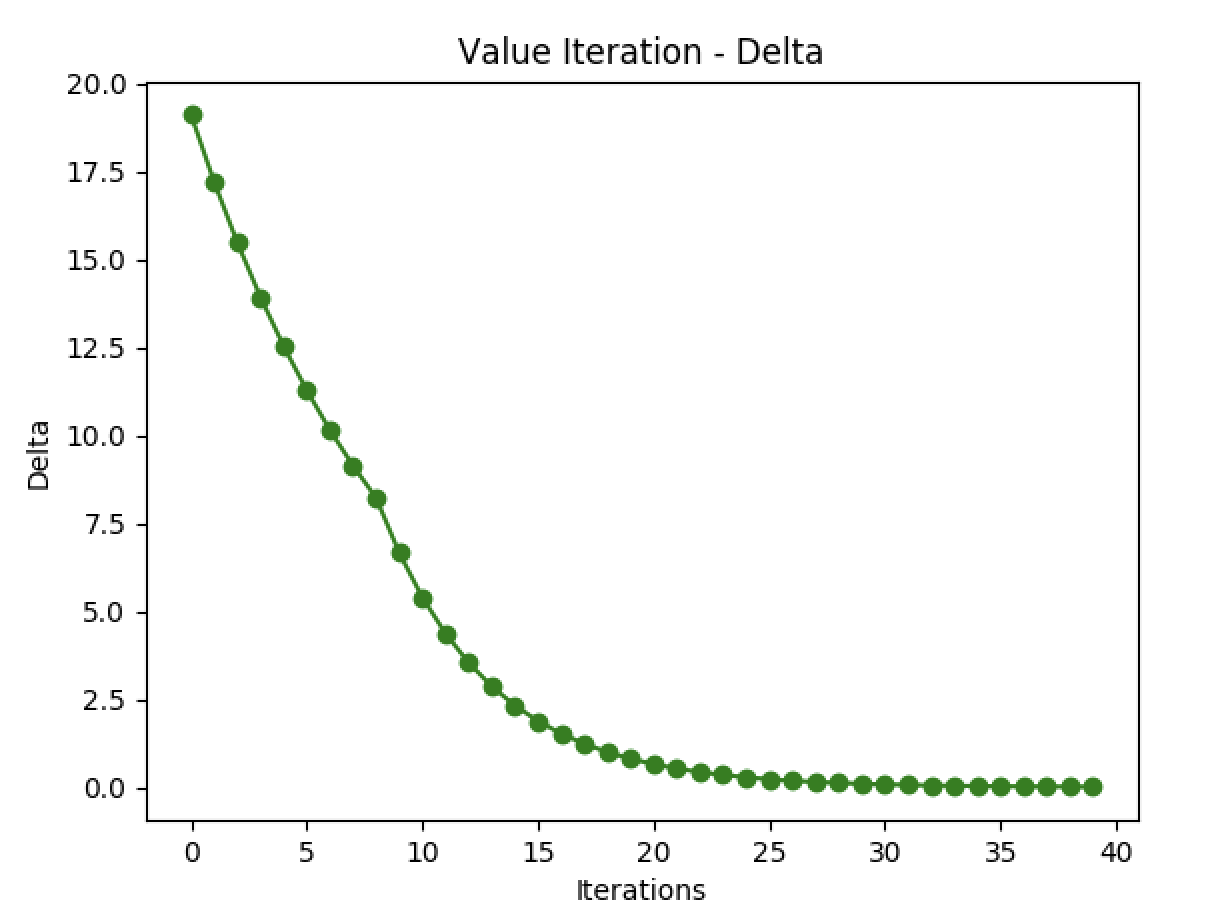
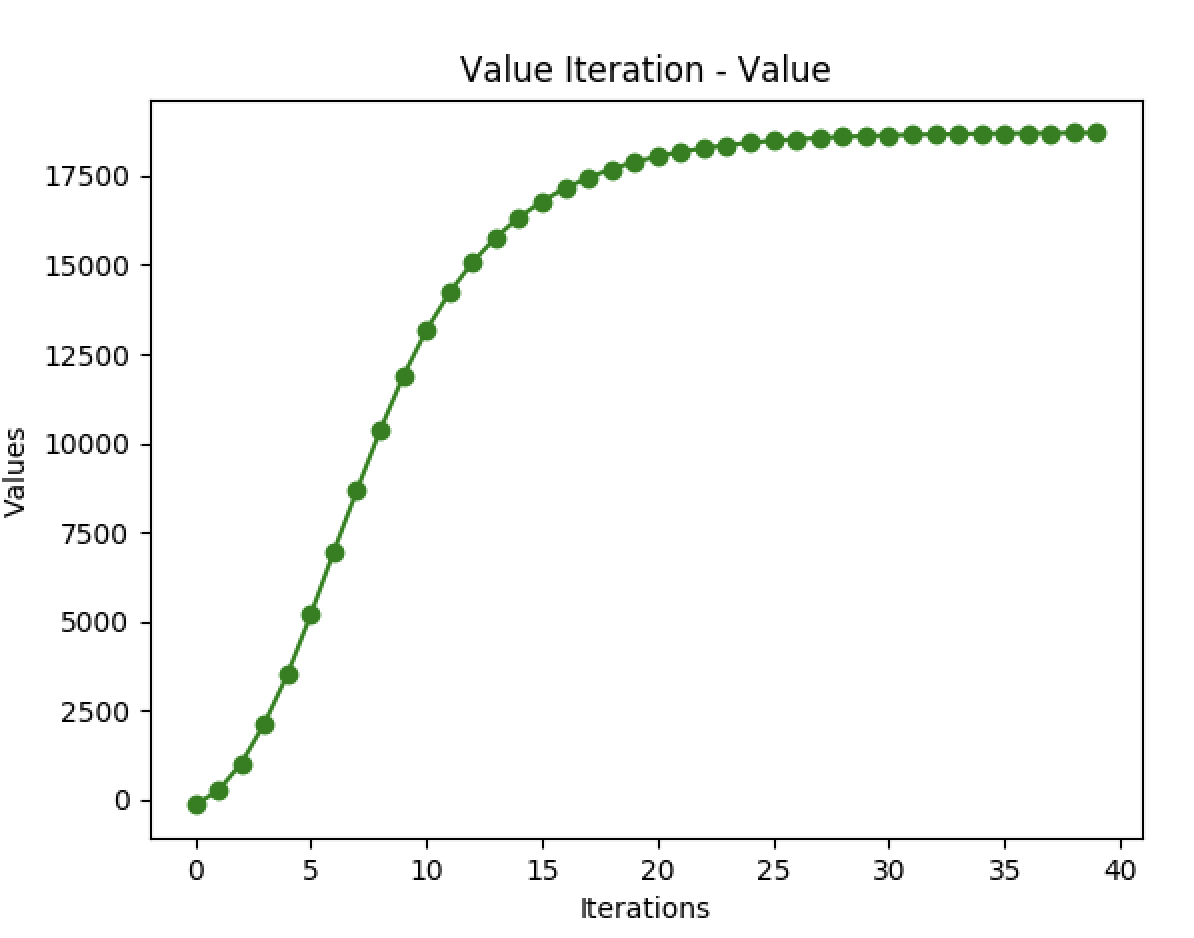
Moving onto the taxi problem, I ran the same analysis as I did for the frozen lake. First, I wanted to understand how different values for the discount factor might impact how long it takes for value iteration to converge. As shown in the table below, the taxi problem actually does not require a higher discount factor, possibly because it is a deterministic problem space, not a stochastic one. This means that each time it chooses a given action, the result and reward is the same. The policies selected by all three of the below runs were the same.

|  |  |  |  |
| --- | --- | --- | --- |
| **Theta = 0.0001** | **Discount = 0.9** | **Discount = 0.99** | **Discount = 0.999** |
| # of iterations required | 62 | 609 | 6079 |
| Processing time | 0.65s | 6.63s | 70.2s |

Moving on to analyzing theta, I chose a discount factor of 0.9 since in the last experiment, it didn’t make a difference to choose a larger discount factor. Interestingly, it also didn’t make a difference whether or not I used theta, since the problem converges quickly enough to result in the same policy regardless of which theta value I chose in the below range. Also, since I initialize value iteration with random values, it’s interesting to note that theta reaches 0.0001 quickly in select cases.

|  |  |  |  |
| --- | --- | --- | --- |
| **Discount = 0.9** | **Theta = 0.01** | **Theta = 0.001** | **Theta = 0.0001** |
| # of iterations required | 40 | 51 | 40 |
| Processing time | 0.92s | 1.50s | 1.07s |

As a result of the two analyses above, I used a discount factor and a theta of 0.01 to get to a final value iteration policy. As you can see below, convergence does indeed happen much more quickly in this scenario in terms of number of iterations. Further, you don’t see any stochastic blips as you did before in the frozen lake problem when observing the policy changes graph below.

*Policy Iteration applied to the Taxi Problem*

Next, I wanted to apply a similar methodology to the policy iteration algorithm. Below, the table shows a number of different runs with different discount factors. As expected, the # of iterations required to converge is lower in policy iteration than with value iteration. Also, the impact of discount factor on the overall resultant policy is minimal, as it was with value iteration. Again, this is likely because the model here is deterministic, so convergence happens quickly. In all the runs below, none of the resultant optimal policies were different across the different discount factors. Interestingly, as the discount factor increased, the # of iterations required decreased. This is different from the frozen lake case, where as the discount factor increased, the # of iterations required increased as well. Again, this is likely because the taxi model is not stochastic, so it does not need as high of a discount factor in order to learn the different states and actions.

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Discount = 0.9** | **Discount = 0.99** | **Discount = 0.999** |
| # of iterations required | 11 | 10 | 7 |
| Processing time | 5.99s | 44.89s | 309.26s |

In this analysis, we also discover that policy iteration, although requiring fewer iterations, takes longer to complete because each iteration is computationally more complex.

*Value Iteration compared to Policy Iteration in the Taxi Problem*

Ultimately, our results showed that value iteration and policy iteration result in the same optimal policy once they converge by comparing the policies chosen by both algorithms. However, value iteration is a faster algorithm, although requires more iterations. Policy iteration requires fewer iterations, but each iteration is significantly more computationally expensive.

*Comparing the Frozen Lake to the Taxi Problem*

In both cases, policy iteration and value iteration had similar characteristics in terms of how they responded to changing parameters, and also found the same optimal policy. Both were also able to converge to an optimal policy. Further, policy iteration required fewer iterations but took a longer time to calculate, whereas value iteration took more iterations but took a shorter time to calculate.

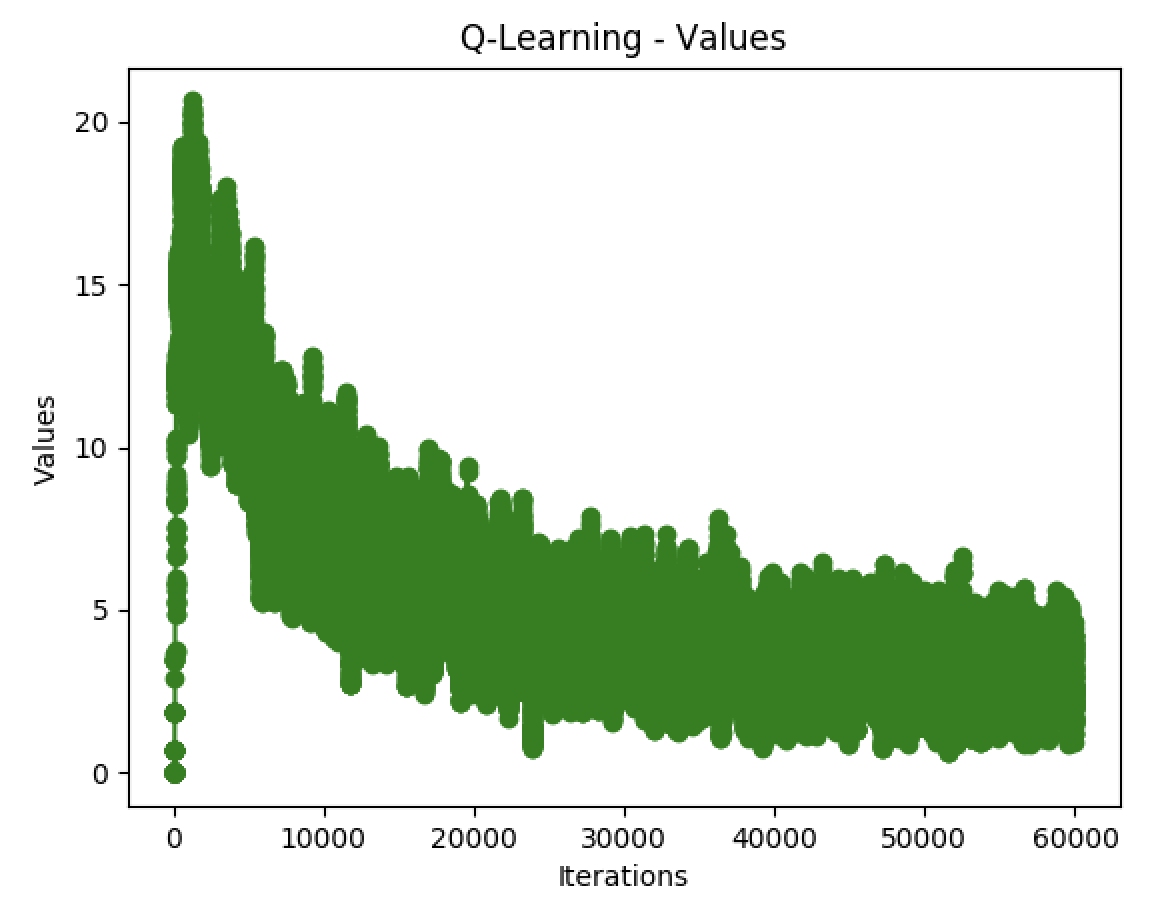
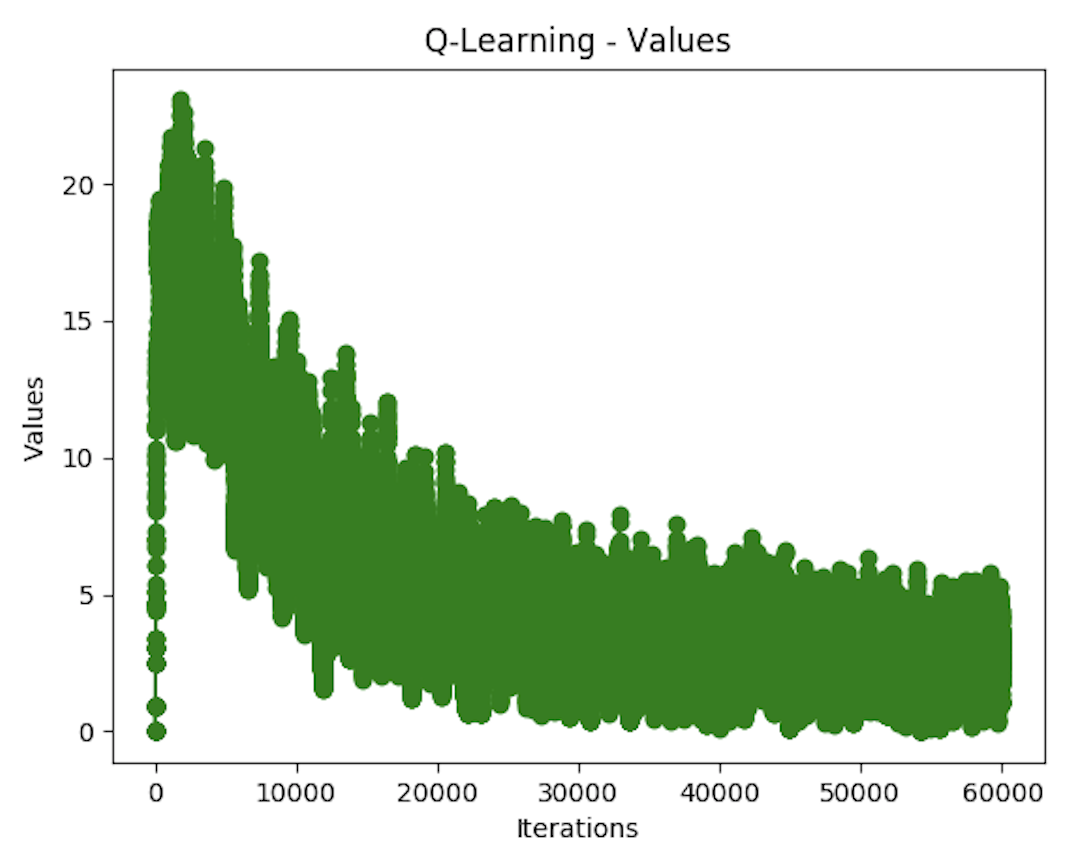
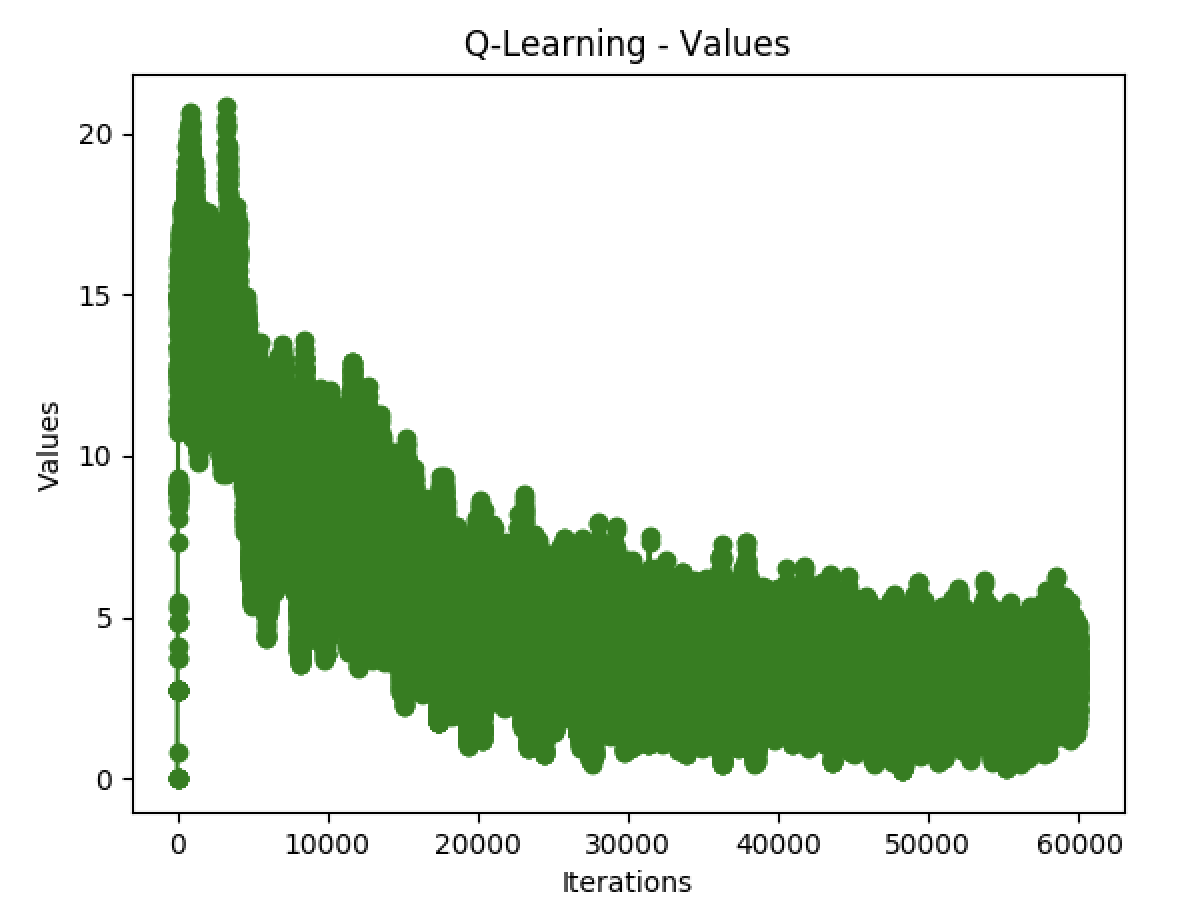
However, there were interesting differences between the two as well. Given the stochasticity in the frozen lake problem, it benefited from more iterations prior to convergence. However, since it did have fewer states than the taxi problem, the taxi problem took a lot longer to run each algorithm, since it was iterating through all the states. So, the number of states increased the amount of computing time, whereas stochasticity increased the number of iterations required to converge.

*Q-Learning*

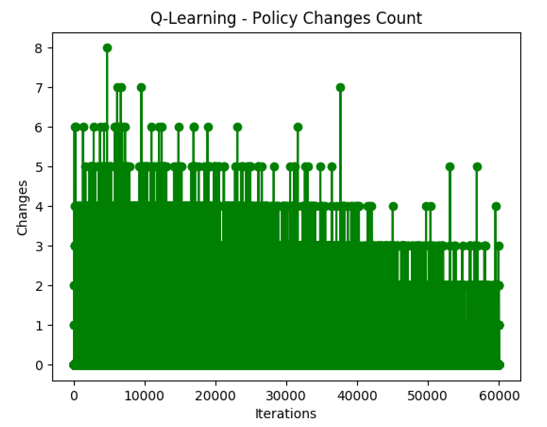
Moving onto Q-Learning – I wanted to test how the different characteristics of each dataset would impact the efficacy of Q-Learning. Q-Learning is interesting in that it doesn’t require a prior knowledge of the model – instead, the learner runs multiple runs in an effort to observe and learn what the model looks like. My implementation of Q-Learning leveraged epsilon decay in order to introduce randomness in how the learner explores different states.

*Q-Learning with Frozen Lake*

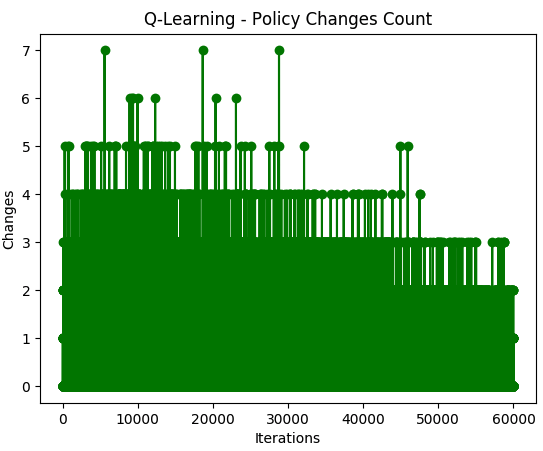
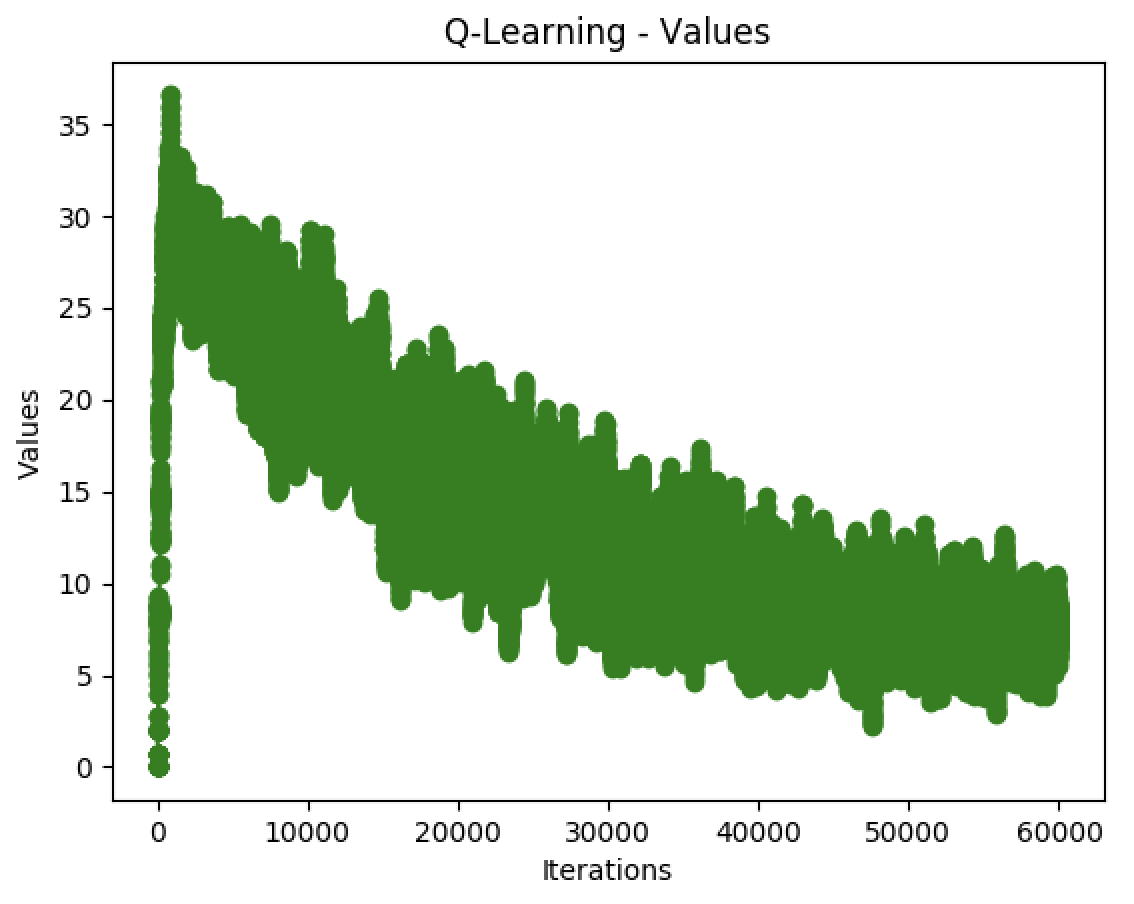
Similarly to how I tested value iteration and policy iteration, I wanted to toggle the learning rate, gamma, and epsilon. The learning rate impacts how quickly the policy adjusts to new information. Gamma impacts how forward looking the algorithm is, and epsilon toggles how often the algorithm chooses a random action. Given that the results selected by each Q-Learning algorithm did not significantly differ with different values of alpha, I chose to go with 0.7 since the processing time was faster at 27 seconds.

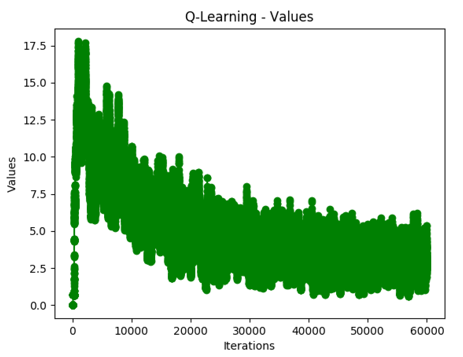
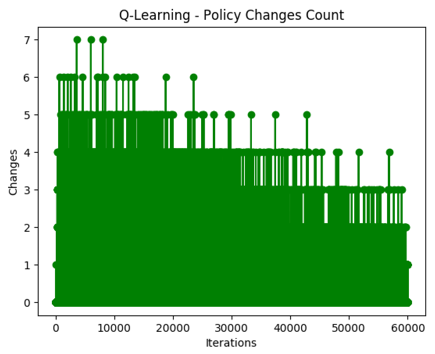
It was also difficult to get the model to converge. In the iteration where alpha = 0.7, the model was still changing the policy even at 60,000 iterations.

My suspicion was that because the frozen lake problem is stochastic, it is converging more slowly. Every time it sees a random event, it disproportionately impacts the optimal model.

To fix this, I tried a different value for gamma, which essentially changes the weight of future values. Previously, my gamma was 0.9. In order to make future values matter more (and hence reduce the impact one outlier has on overall results), I increased gamma to 0.99. This ultimately did not improve convergence, and actually resulted in longer run times (increasing to 36 seconds).

Reverting back to gamma of 0.9, alpha of 0.7, I wanted to see how epsilon might impact performance. My guess was that for a stochastic model like frozen lake, it’s better to have a higher epsilon so that we can adequately explore the space.



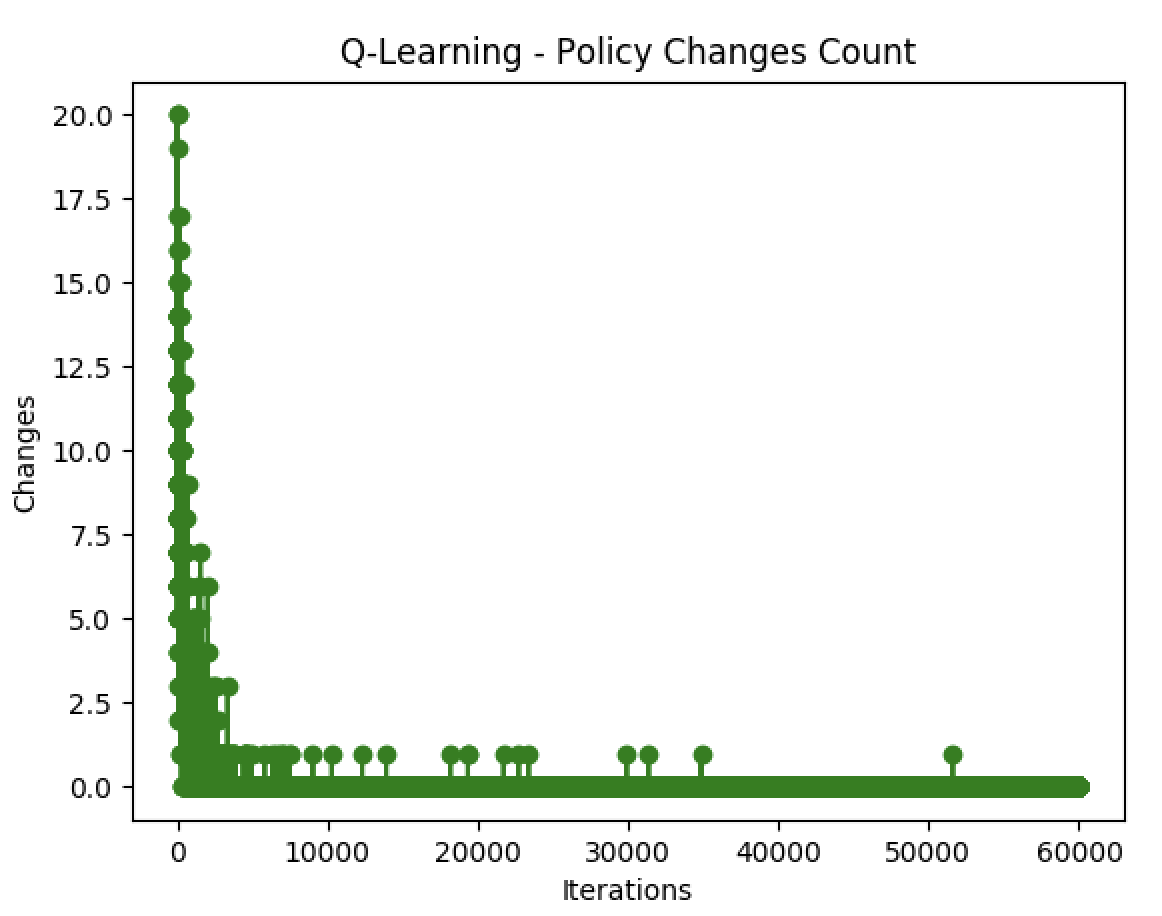
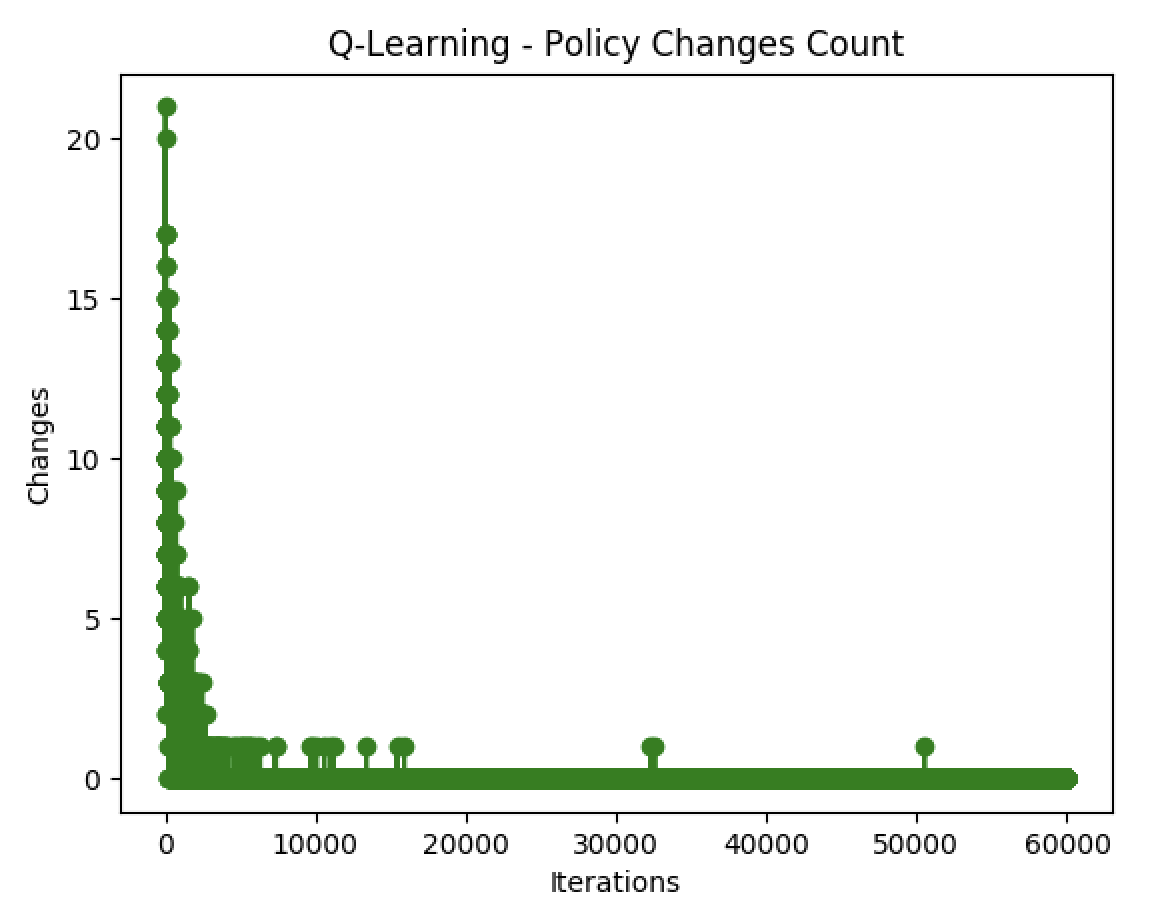
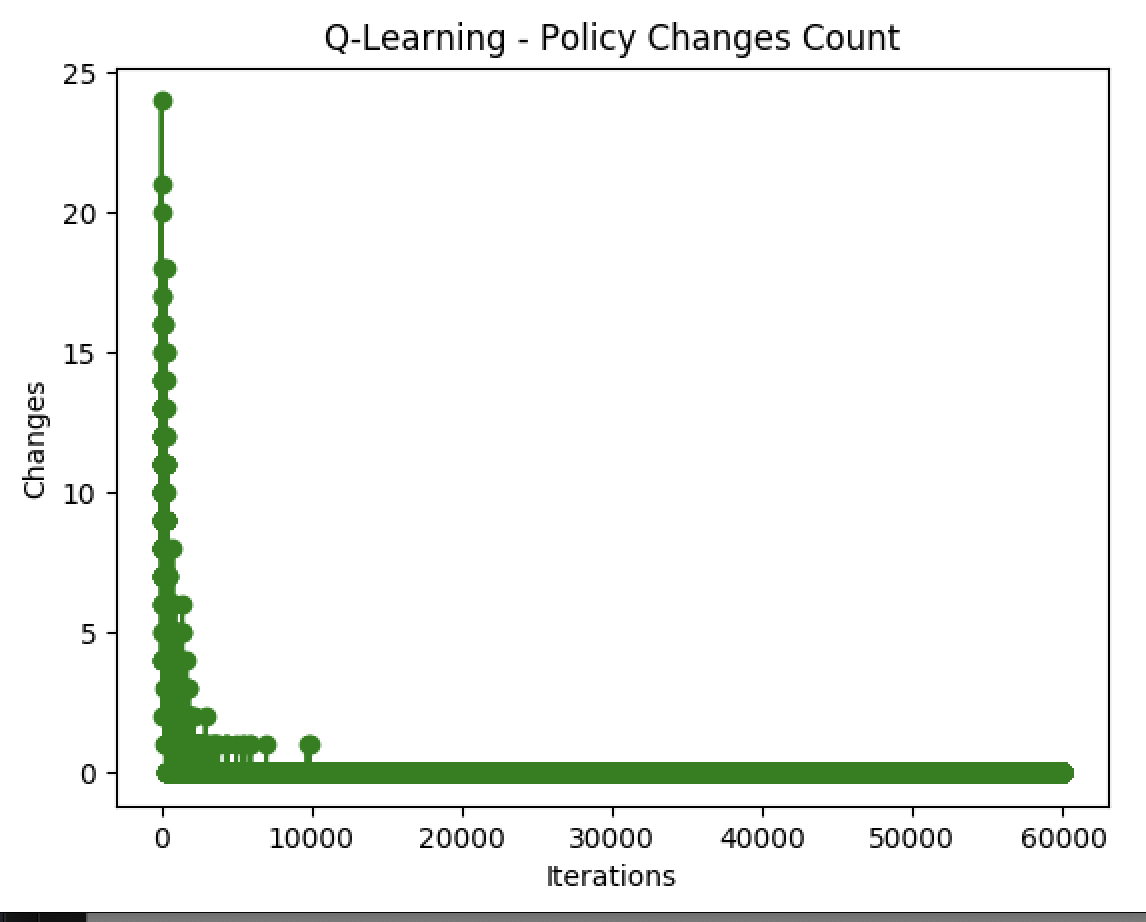
As expected, the model actually converges slightly faster than when epsilon is higher. However, the model chosen by Q-Learning with epsilon of 0.99 is different than that chosen by 0.9. Ultimately, Q-Learning has trouble converging and does not handle the stochastic nature of the frozen lake as well as value iteration and policy iteration does.

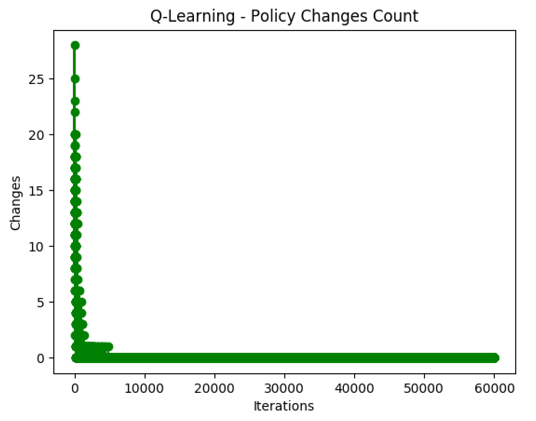
*Comparison of Q-Learning with Policy / Value Iteration for Frozen Lake*

Ultimately, Q-Learning takes more iteration and more time to find an optimal policy than policy and value iteration. By comparing the final configured models with each other, I discover that Q-Learning also chooses a different optimal model than policy and value iteration. The difference is small. Only one state chooses a different action. However, it does show how Q-Learning has more trouble converging on an optimal policy.

*Q-Learning with Taxi Problem*

Moving on to the taxi problem, I apply the same steps that I did for frozen lake to the taxi problem. First, I compared different values of alpha (0.7, 0.8, and 0.9).

Based on the graphs, an alpha of 0.9 results in better convergence over the iterations. However, the policies are quite different from each other – ranging from being different into 16-26 states. I also tried to see if increasing gamma would help, but like in frozen lake, doing so did not make a significant difference in results. Finally, I tried decreasing epsilon. This actually helped the policy converge more quickly. When comparing the policies between epsilon 0.99 and 0.9, the actual policies were actually quite similar, with only 9 states that were different.

By comparing the policy achieved with a 0.9 alpha and a 0.9 epsilon, the policy generated is only different from the value iteration policy in one state.

*Comparison using Q-Learning for Frozen Lake and Taxi Problem*

Ultimately, tuning the parameters appropriately resulted in policies that were very similar to the policies found by value iteration and policy iteration for both the frozen lake and the taxi problem. However, getting there took a lot longer and also required a lot more iterations.

Comparing between frozen lake and the taxi problem for Q-Learning, it’s clear that the stochasticity of frozen lake means that more time needs to be spent on learning, convergence is slower, and results fluctuate more. On the otherhand, the deterministic qualities of the taxi problem allow it to converge more quickly, despite having more states to explore.

*Conclusion*

In this assignment, I applied value iteration, policy iteration, and Q-Learning with epsilon decay to two MDP problems. I discovered that for both MDP problems, value iteration and policy iteration were more effective, stable algorithms with which to discover an optimal policy. Q-Learning allows for more flexibility if an environment is not well understood, but since we knew the rules our of environment in both MDP cases, it resulted in strictly worse policies. However, Q-Learning was able to converge quite quickly, despite slight fluctuations in the optimal policy, impacted by the stochasticity of the frozen lake environment. Despite this, Q-Learning was able to find a policy close to the optimum policy for both MDP cases. However, Q-Learning did take a much longer time to train, so it should really only be selected in cases where the environment is unknown.

1. <https://github.com/openai/gym/blob/master/gym/envs/toy_text/frozen_lake.py> [↑](#footnote-ref-1)
2. <https://github.com/openai/gym/blob/master/gym/envs/toy_text/taxi.py> [↑](#footnote-ref-2)
3. <https://www.kaggle.com/angps95/intro-to-reinforcement-learning-with-openai-gym> [↑](#footnote-ref-3)