Markov Decision Processes

Branden Kim

bkim400@gatech.edu

I. Introduction

Markov Decision Processes are a framework of a type of problem that can be solved utilizing Reinforcement Learning Techniques. The defining characteristics of these problems are that they all share the same setup in their problem: states, actions, transitions, and rewards. Furthermore, a fundamental underlying principle within Markov Decision Processes is the Markovian Assumption which assumes that a current state only depends on its first previous state and doesn't consider the entire past history of states. This paper aims to analyze and dive deeper into MDP Problems and utilize Reinforcement Learning techniques in order to solve for an optimal playing strategy within those problems. The paper aims to understand and analyze differences in the problems and how the underlying nature / rules of the game affect the way the RL algorithms work.

II. MDP Problems

To compare and contrast, there are two MDP problems that I have chosen to analyze the different RL techniques. The first game is Blackjack, the classic card game where the goal is to get as close to 21 as possible. The second game is a game called Frozen Lake which is a 2d Grid World game that focuses on a player getting from a start to a goal cell in the grid world. Two caveats in the Frozen Lake game is that there are cells that are holes which lead to penalties if landed on, and when moving there is a stochastic probability that you move in an adjacent direction due to the slipperiness of the ice.

The reason for choosing these MDP problems are, first the underlying rules of the game are simple enough such that the number of actions and what type of actions that can be taken can be easily understood. This makes the analysis of the underlying RL algorithms much easier as there are less variables / elements to have to juggle (I want to start off my journey in RL with simpler problems in order to gain a better understanding of the fundamentals). Second, both of these MDP problems contain elements in the domain space of the Game World that I am interested in for further research / experimentation that are analogous to other kinds of possible MDP problems. One of my goals is to make an RL agent for super mario games which are not Grid World problems but have discrete action sets similar to that of the Blackjack game. I am also interested in utilizing RL for building an agent for Chess which is pretty analogous to Frozen Lake because it is a 2d grid, each move contains some probability of likelihood from the enemy, and there are possible moves that are not desirable as it can lead to losing the game pretty quickly. These two MDP problems thus set up the foundations for applying the RL techniques to more challenging but analogous MDP problems that I am interested in, assuming I can understand and analyze the RL techniques. Furthermore, the nature of the two domains, Blackjack and Frozen Lake, are quite contrasting as one is a grid world and the other isn't and one contains probabilistic actions while one is deterministic. I hope that the differences in domain problems lead to different discoveries in the analysis of the RL techniques that allow me to gain a deeper understanding.

III. Model-Based Approaches

The two model-based approaches that we learned / covered in lecture were Value Iteration and Policy Iteration. Both approaches are very similar other than the fact that in Value Iteration, you try to calculate the max utility per action at each state's possible actions whereas Policy Iteration ends up starting off with a random policy and utilizing that policy to guide selecting an action. As the policy iteration continues, it updates the policies by what action it should take by calculating the Q values or utility values per action in each state and taking the best one at that particular time. Both of these approaches work well due to the fact that they have a "true" model that defines the transition probabilities and the rewards that are received when making an action from any particular state. This means that a global maximum convergence point is always reachable (won't get stuck in local minimum) for solving RL problems with these model-based approaches. That means we need a different sort of criteria for understanding how well the RL Model-Based technique works on our MDP problems.

IV. Model-Based Measurements

In order to measure some form of a score or "goodness" criteria for these model based approaches, the several following measurements have been chosen. We are going to look at different Model-Based approaches converges with respect to each MDP

problem, using the Mean Value with respect to number of Iterations as my main approach. I decided to make this the main criteria for determining convergence as with the model-based approaches because we know the true rewards and transition probabilities from any particular state and any particular action. Mean Value is a very good measure because once the RL technique converges, that means the Mean Value shouldn't really change as the Value utilities per state have propagated out from the reward states. Because the Model-Based RL techniques are guaranteed to converge to the true optimum, there are not many other measures of convergence that we can really measure (there's no real point since it is going to converge to the optimum). Things like Cumulative Reward don't matter in this scenario because the rewards from going from one state to another off an action are already their "true" values and thus it is not like the RL technique is "learning" these reward values. With the Mean Value, we can find how many iterations it takes for each MDP problem to be able to reach convergence based on the state size of their respective problems. For Blackjack, I ended up not modifying the state space as I thought it wouldn't be a fair comparison as the underlying game contains strategies based on the fact that certain cards combine values to reach the threshold value of 21. If this max score were increased to a higher value such as 50, a lot of strategies towards winning Blackjack change as the combinations of cards to reach a value of 50 change since with a higher threshold, there is a viable strategy to just play in the middle value around 25/30 since the probability of the dealer getting a value greater than that is lower depending on what cards come out. However, for the Frozen Lake problem, I varied the state space / problem size greatly as it ends up being the same problem just at bigger sizes and the winning strategy should end up being the same. Lastly, I will provide a Policy Map of what actions should be taken at which state to compare the resulting policies from each MDP problem and we will see if they are optimal given that each problem has reached convergence. As for my prediction, I think that for comparable state space sizes of each problem, Blackjack will reach convergence quicker with Value Iteration and Frozen Lake will reach convergence with less iterations with Policy Iteration. This is because Blackjack needs to consider policies for each possible state based on the possible utility value that comes out from that action. It would be quicker to just calculate the true utility values at each state regardless of what policy you end up taking as you might end up spinning in loops longer with policy iteration. Frozen Lake on the other hand would do better with policy iteration as due to the stochastic nature of the actions, calculating all possible probabilities and their resulting utility values and which is the maximum would take very long for every state, especially as the state space gets larger. Policy iteration on the other hand, can start with a random policy action and as iterations increase, the policy it takes quickly becomes more and more optimal as you wouldn't take actions that clearly lead to worse results from the current policy you have.

V. Model-Based Results

The results for Blackjack followed the hypothesis I made fairly accurately.

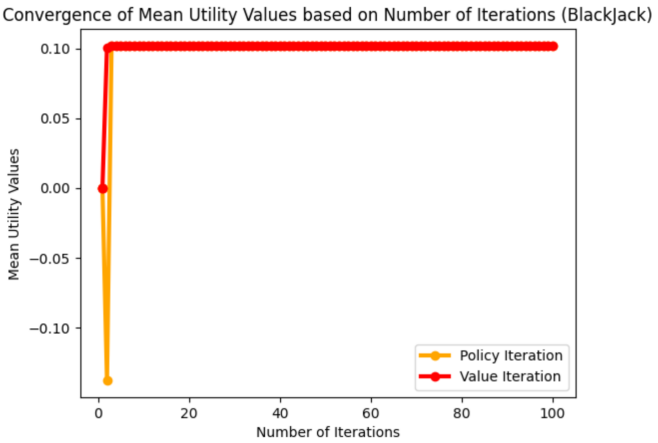


Fig. 1 Mean Utility Values based on Number of Iterations (Blackjack)

Looking at the figure above, we can see that Value and Policy Iteration converged rather quickly in terms of the number of iterations for the Blackjack MDP problem. This is interesting of why this happens but it makes sense for the MDP problem domain of Blackjack as each action you can take from a particular state is deterministic and the number of actions is small (2) which is just getting another card or staying where you are at. For the context of Model-Based problems where the true rewards and transition probabilities are known, this makes converging to the optimal policy very quick as at this point it is just as a matter of propagating values and calculating utilities based on the true probabilities. For both Value and Policy iteration, I noticed that it converged after about 2 iterations. As I hypothesized in the previous section, the Blackjack MDP problem performs better with Value Iteration rather than Policy Iteration.

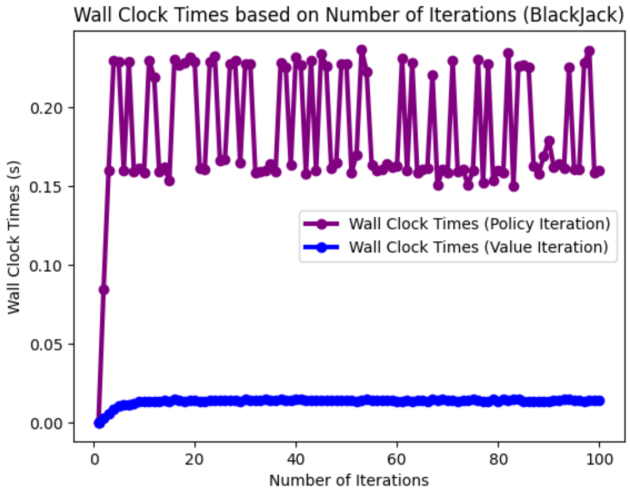


Fig. 2 Wall Clock Times based on Number of Iterations (Blackjack)

As we can see from the figure above, once convergence is reached, the wall clock times (performance) of Policy Iteration is much higher than that of Value Iteration. The reason for this I think is exactly what I outlined in the previous section. Using policy iteration, Blackjack needs to try a bunch of non-optimal actions in order to start finding the optimal actions so there is a bunch of going back and forth. However, with Value Iteration, since we have the true probabilities, it is just a simple calculation of each possible action at each state and propagating the values based on the probabilities. At each iteration of the loop, there is no waste because it is calculating the true value for each state at every iteration of the loop. If we take a look at the policy map for both model-based approaches, we get these results.

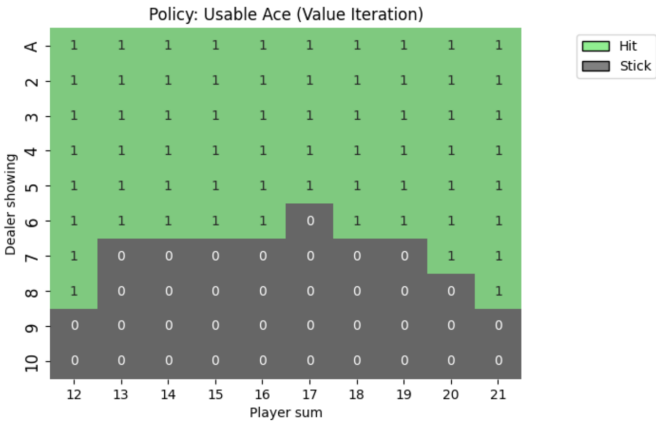


Fig. 3 Policy Map for Value Iteration

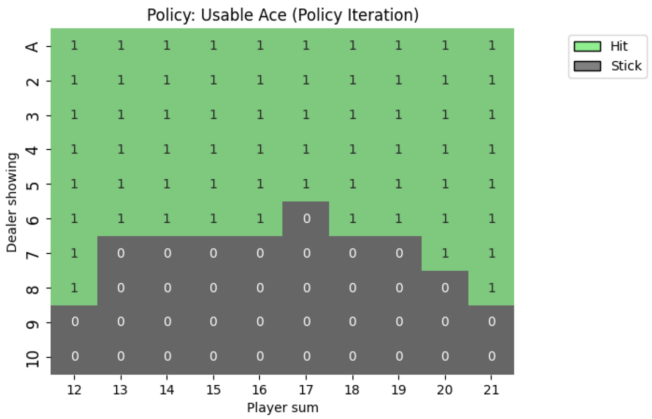


Fig. 4 Policy Map for Policy Iteration

From the figure we can see that both Policy and Value iteration end up getting the same optimal policies which means that we are getting what we expect with both of the model-based algorithms. However, it is interesting to note that this doesn't really seem to be an optimal policy for playing blackjack as the player's decision to hit or stand is based off of the fact the dealer has a small hand, not the fact that the player has a high card number. I think this is because the P matrix that I utilized (came with betermdptools) was actually not the "true" transition probabilities matrix. After looking at all of the code / conversion of observation states to a single discrete value function, the probabilities matrix being incorrect was the only thing that I could think of.

Now let's take a look at the frozen lake example.

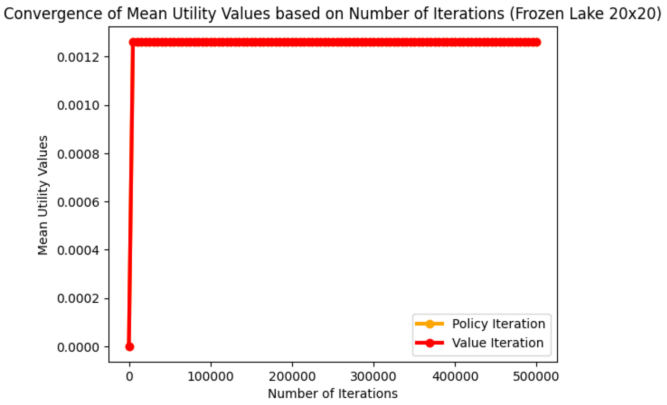


Fig. 5 Mean V vs Number of Iterations Frozen Lake (20x20) Grid

Above we can see the results of running Value and Policy Iteration for a fairly large state space (400 is my control) and it looks like it converges fairly quickly. This is to be expected similar to that of Blackjack's graph as we are running a Model-Based approach where we know the true probabilities. In this MDP problem I wanted to explore trying different state space sizes. Here is the result for 8x8.

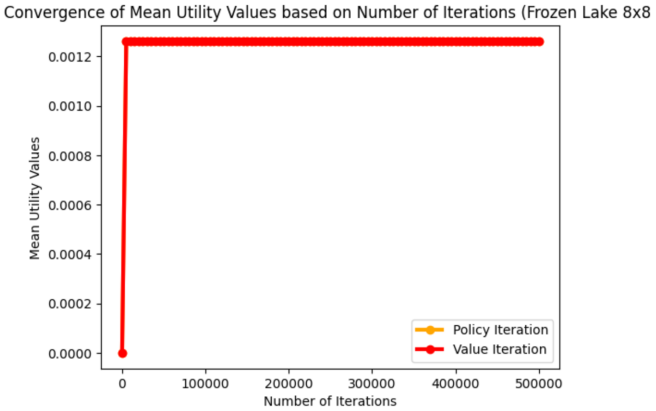


Fig. 6 Mean V vs Number of Iterations Frozen Lake (8x8) Grid

Similarly for a 32x32 Grid.

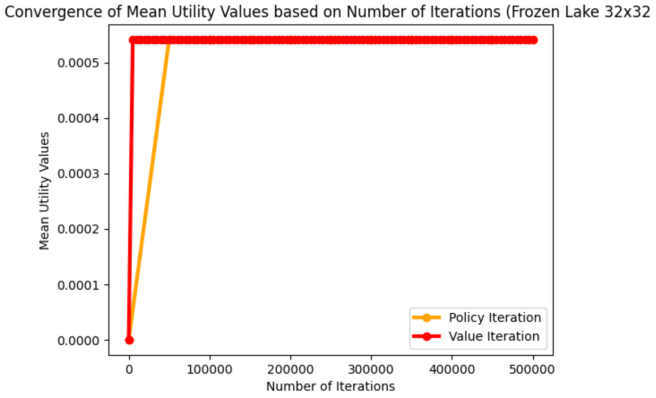
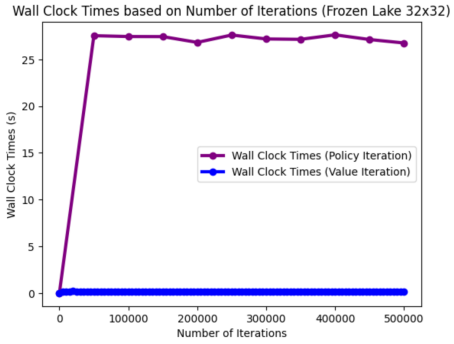
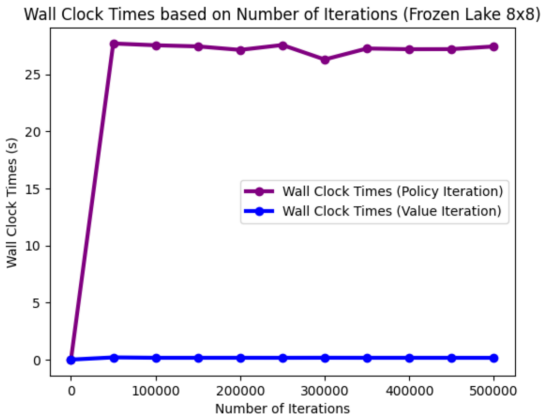
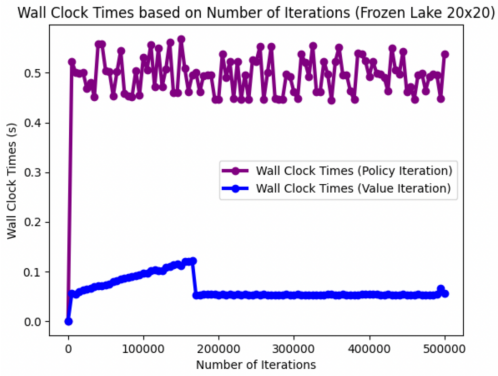


Fig. 7 Mean V vs Number of Iterations Frozen Lake (32x32) Grid

For each of the state space sizes, the actual number of iterations it took to converge didn't really change too much, although the 32x32 grid had to have some modifications because it took way to long at high iterations (which is expected because policy iteration is very slow with a large number of states and episodes). However, the wall clock times increase as we expect with higher state space sizes.







Not surprisingly, as the state space gets larger on the MDP problem, the wall clock times increase at a very high amount. It is interesting to see that Policy Iteration takes so much longer than that of Value Iteration for the Frozen Lake MDP Problem. Unlike my initial assumption that Policy Iteration would have performed pretty well on Frozen Lake, it seems that in general Policy Iteration takes much longer than that of Value Iteration. This might be because in general Policy Iteration takes a random starting initial Policy and then makes its actions accordingly, which can lead to loops of iterations of taking bad actions before getting to the optimal policy.

VI. Model-Based Results

Overall, when running the agents after they have converged from using Value Iteration and Policy Iteration, I tried testing them on actual training examples and seeing how well they did. When testing it on Blackjack, the model did not do very good, where the value iteration resulted in losing more often than not as it resulted in a negative score. However, when it utilized Policy Iteration, the testing score went up as it got a majority of the plays correct. This is surprising because initially I would have assumed that Policy Iteration and Value Iteration would have performed at a similar clip as they are both finding the optimal policy from the model-based transition probabilities.By contrast the

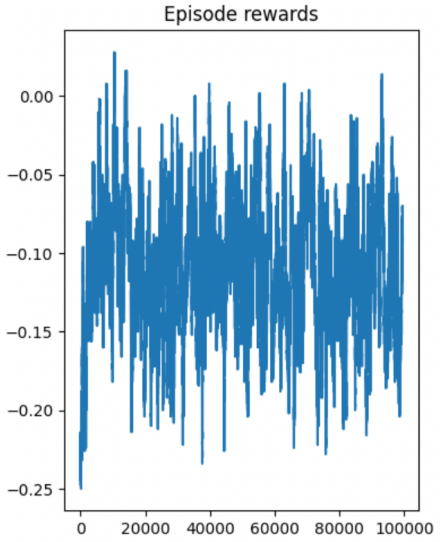
Frozen Lake did very well with the results when running on the testbed. Both Value Iteration and Policy Iteration were able to receive pretty high scores. I am not sure why this MDP problem was able to do so, but part of it is because I am assuming that the probabilities matrix is more accurate for the Frozen Lake problem compared to the BlackJack problem. Furthermore, the scores varied by the state size. The smaller the state size, the greater the test scores. This follows my intuition because a smaller amount of states means that there needs to be less amount of iterations to reach convergence. I kept the number of episodes the same for each state size and I noticed that the bigger states tended to have worse scores because the optimal convergence values might have not been reached for the hyperparameter values I was utilizing.

VII. Model-Free Techniques

Another method of going about learning the optimal policy in a RL setting is to utilize a model-free based approach. In this example, instead of having to utilize a model that perfectly describes the situation, you have no knowledge of what is optimal in terms of the rewards you might receive or the probabilities of the transitions you might take based on an action. To get around not having the "true" answers, an approach is to iteratively sample the data for the rewards as well as the probabilities for transitions in order to utilize that sample as the "true" probabilities when trying to learn the optimal policy. The algorithm that I utilized to sample these random variables is Q-Learning and applied it to both MDP problems. Unlike the model-based approach, since we are taking samples and having to explore the space of taking different actions at each state even though they may be less optimal to us at the time. This is because we actually don't know the "true" rewards we receive from transitioning off of a state with an action and also we don't know the "true" probabilities of going to a particular state off of an action. Now the analysis and how well the Q-Learning algorithm will work for both of the MDP problems change a little bit. Intuitively, we would think that Q-Learning would overall run much quicker and converge quicker for that of Blackjack compared to Frozen Lake. This is due to the fact that the transition probabilities for Blackjack is guaranteed as well as the fact that it has a smaller amount of states compared to the Frozen Lake. It is kind of hard to intuitively guess the difference in convergence between the two MDP problems, but I will hypothesize that it'll take longer for the Frozen Lake to converge. Furthermore, I hypothesize that the results of the scores on the testing examples will be better using Q-Learning compared to that of the Model-Based approaches. Due to the incorrect transition probabilities and how fast it converged, I am skeptical that the model-based techniques retain a high level of accuracy. Thus, even though it is trained on comparable number of episodes, I think the randomness of searching that the Q-Learning algorithm utilizes will actually allow it to gain a better score overall compared to the Model-Based Approaches. In the Q-Learning approach I utilized the epsilon-greedy exploration. Intuitively, it seemed like the choice that made the most sense and doing something like random-restarts took way too long on the Frozen Lakes example so I decided to add the randomness in another way.

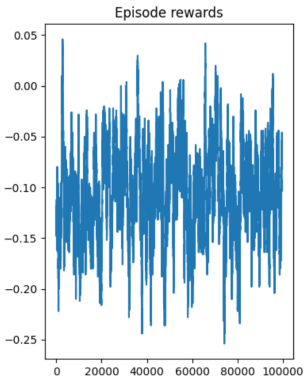
VIII. Model-Free Measurements

For Blackjack, I received pretty good measurements and was able to gain some insight into the Q-Learning Algorithm. For Blackjack, I decided to look at how the reward values change over the iterations as well as a Policy Map and how it fairs with the test environment to see how it performed.

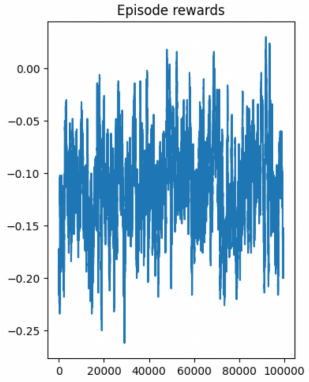


From the screenshot above, we can see that based on the number of iterations, the episode rewards amount increased and then started to vary across a certain amount. From my intuition, I interpreted this as the Q-Learning algorithm converging as it meant that as we increased the amount of iterations, the reward values per state -> transition pair didn't change in a linear fashion. Since the Q-learning algorithm samples the rewards from existing data pairs and then adjusts them based on the action / policy being enforced, if the algorithm didn't converge then we wouldn't see a pattern with the rewards on as the number of episodes increased. To

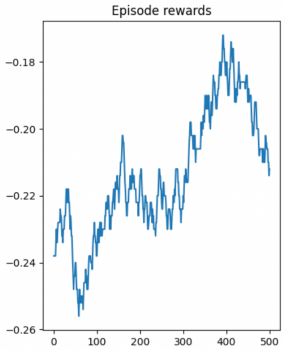
compare this result, I increased the epsilon decay amount and also increased the learning rate by a significant amount. Our reward vs iteration now looks like this.



While they have the same shape, we can see that there are no upward trends in the early parts of the iteration and the variance / variability is much higher with these hyper parameters compared to the previous ones.

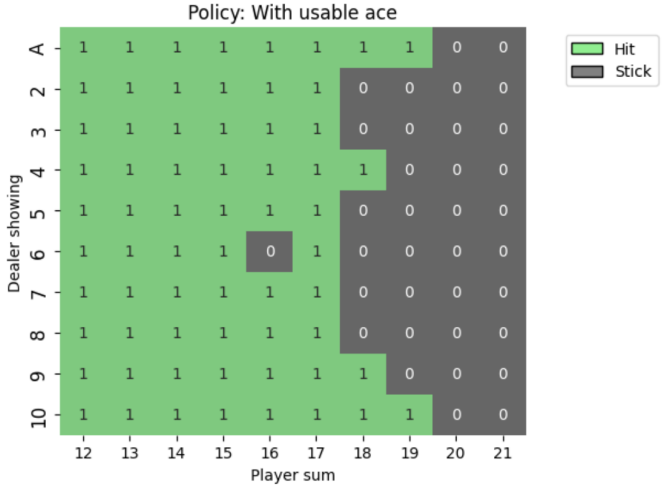


This is after decreasing the learning rate to 0.0000001 while keeping the epsilon value at 0.9. I thought the variability would decrease a significant amount and start to approximate a log line or linear line, but it still has high variability. This suggests to me that maybe due to my high amount of episodes of learning, regardless of whether the hyperparameters change, the Q-Learner is still able to reach convergence.



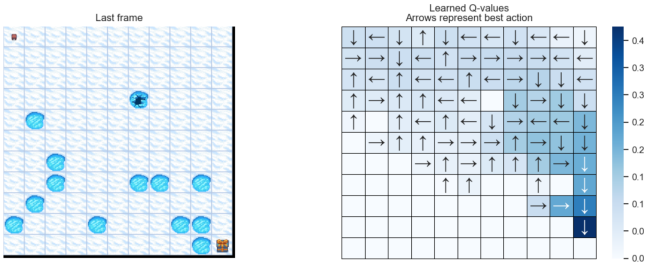
Changing the number of episodes down a significant amount gets the variability that we expected from the Q Learner. We can see in the graph that the reward amount is not nearly the same as the converged graphs (still learning the true reward values as they are all negative) and there is a high amount of variability in the graphs.

Utilizing our previous hyperparameter values, we can get a policy graph as such.

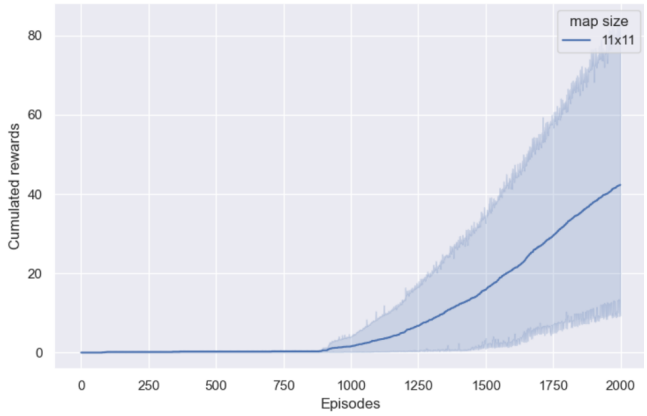


Comparing this policy graph to the one we received with Value and Policy Iteration, this is much more accurate and intuitively makes sense.

Now looking at Q-Learning for the Frozen Lake example, I decided to utilize Cumulative Rewards vs Iterations and the Policy Map to determine criteria for Convergence. For the Frozen Lake example Cumulative Rewards works well because it is able to show the point at which the rewards stop changing and start to increase linearly when cumulatively added. This is actually the point of convergence. For the Frozen Lake MDP problem, I changed the reward system such that when transition between regular tiles, there is 0 reward. Reaching the goal gives you +1, and falling into a hole gives you -1. This means that when the “true” reward values for each tile is not known and we are just sampling there will be variability around 0 because a portion of the tiles are holes which add -1 and there is a goal state which adds 1. However, when convergence is reached, the learner should be able to avoid the holes as much as possible and the cumulative rewards should start increasing linearly because the learner is able to reach the goal state a majority of the time (since it is stochastic).



This is the final policy map I ended up with but I had to reduce the state space size to 11x11. I couldn't end up reaching convergence on the 20x20 space even with 500,000+ episodes. I am not sure if it is due to the amount of holes that I picked, but when looking at those maps, I had a very very high number of ice holes. Even when I changed the probability amount it didn't seem to help as much. However, when going down to 11x11 I was able to reach some form of convergence. We can see in the policy map above that due to the high number of holes in the lower left hand side of the grid, the learner actually doesn't go in that direction and decides to mainly stick to the upper right hand side.



Here is the cumulative reward vs number of episodes map. We can see that around 1000 episodes, the cumulative rewards start to go in a linear upward trend, indicating that the learner has converged as I explained a bit earlier.

IX. Model-Free Results

It is interesting how the Q-Learner's policy map and scores when run against the test environment were so much higher than that of the Model-Based approaches. I still don't know the exact reason why this occurred but in the back of my mind the transition probabilities matrix being incorrect for the model-based approaches seems pretty likely. Furthermore, when utilizing the Q-Learner approach I was able to vary the learning mechanism by utilizing some exploration instead of exploitation in the form of the epsilon-greedy exploration. This added amount of randomness might have gotten the learner into situations where it was able to try a new, different action and calculate higher utility values based off of that different action. From the graphs on the rewards for Blackjack I included before, it seemed to be that changing the values of hyperparameters to allow the learner to explore more made some difference but due to the high amount of episodes, it was still able to converge to some amount. It might be that the Blackjack problem domain is too simple to learn and such, adding varying levels of decay for exploration doesn't make a difference as there is really only one optimal strategy that can be learned and the high number of episodes allows the Q-Learner to learn the amount.

For the Frozen Lake MDP Problem, the different hyperparameters and the exploration strategies worked much better as the stochastic nature of the actions added a lot of variability that could be learned from the Q-Learner algorithm. With a small enough state space, the amount of episodes needed to converge was much lower, but it quickly snowballed out of control when dealing with state spaces that exceeded 400+. Unfortunately, I wanted to try the Q-Learner example at higher state spaces such as 32x32 or 20x20, but the algorithm wouldn't converge and also ran into a lot of issues in terms of the running time even if I changed the hyperparameters such that it was able to have a smaller threshold before the convergence criteria was reached.

I am fairly surprised that my Frozen Lake Q-Learner was not able to reach convergence at higher levels of state spaces, and I am actually not that sure what I did incorrectly. I messed with changing the learning rates, epsilon decays to encourage a lot more exploration and yet it still resulted in not successfully learning anything. Part of the reason I think is due to the high number of ice holes in the map and due to that it was just too hard for it to determine a successful path throughout the grid world to the goal. Another thought that I had but didn't have time to explore is changing the reward amount drastically. I changed a bit where I added negative weights toward the ice holes, but since it didn't help much I was exploring other options for being able to reach convergence but it didn't help. I am going to try to explore making the rewards drastically different where the goal is +100, holes are -100, and regular tiles are 0 to see if that helps the Q-Learner be able to learn something.

X. Conclusion

Overall the Q-Learning algorithm for small state spaces worked so much more accurately and led to better results than the model-based approaches which greatly surprised me.