Markov Decision Process

For this assignment we were instructed to explore three different Markov Decision Processes (MPD) across three different environments. There are five key characteristics of all MDPs, those are States, Actions, Model, Rewards, and Policy. The States are all of the possible scenarios within your model. Think of a small 2x2 grid, there are 4 States: (0,0), (0,1), (1,0) & (1,1). The Actions are all possible actions within our environment, back to our grid world we can go Up, Down, Left or Right. The Model is the actual decision maker, and in our case, those are called Value Iteration (VI), Policy Iteration (PI), and Q-Learning (QL), which we will get into later. The Rewards are how we tell the model if it is failing or succeeding, this ranges from negative infinity to positive infinity. Finally, the Policy is the best course of action the Model could come up with, so back to our grid world problem it out be Up then right or Right then Up so we arrive at the top right square.

Explanation of Environments

These two environments, in which we will test the models, can be described as a grid world problem and a non-grid world problem. The first, called Frozen Lake (FL) is classified as a grid world problem. The premise of FL is about traversing a frozen lake that has holes in it and trying not to fall in. The States are an MxM grid ranging from 0->M on the x-axis and 0->M on the y-axis. The starting state is located at (0, M) and the goal is located at (M,M). The actions are what we described above: Up, Down, Left, Right. The Rewards in this environment are only given for reaching the goal or falling into a hole, +1 and -1 respectively, then all the other states have a reward of 0. The Models we have already discussed, and the policy will be derived later.

The second environment is called Blackjack (BJ), and it is exactly what it sounds like. This non-grid world problem has a fixed number of states, which are a combination of what cards are on the table, what two cards I have, what cards the dealer has, and what cards are remaining in the deck with their associated probabilities. There are 2 Actions in this environment, skip and draw a card. The models in this environment are given a reward, be it positive or negative, for a draw, a loss, a 'perfect win' which is getting exactly 21, and a win which is whoever is closest to 21 without going over 21. As for the Policy, we will cover that later.

Now that we have set up our environments, here is how the analysis will be structured. We will start by examining VI and PI within the FL environment on a large and small state size. Then we will discuss the policies and convergence of the 2 models on the different state sizes. Then, we will examine QL in the same 2 environments then cross analyze. For the BJ environment we will first look at VI and PI with a varying reward structure. We will have 3 different types of rewards, Higher Risk, Base, and Higher Reward. The Higher Risk structure is weighted more heavily on punishing the model for making mistakes with draw:0, loss: -10, perfect_win:1.5, and win:1. The base is where all the rewards are basically the same draw:0,

loss:-1, perfect_win:1.5, and win:1. The Higher Reward structure focuses on rewarding the model for doing well and not focusing on the punishment with draw:0, loss:-1, perfect_win:1.5, and win:10. We will then explore QL on these same pretenses then cross analyze.

Expected Challenges

Now, let's talk about the challenges these scenarios give us. First, the biggest challenge comes from FL where there are no rewards except at the end and the holes. This isn't a big problem for smaller sizes like a 4x4 or a 5x5. The goal is still close to the starting point and there are only 25 states max, so it is easy for the algorithm to traverse and learn where the holes are. But, in bigger sizes like a 50x50 you have 2,500 states and the way Q-Learning behaves, it struggles to back propagate the Q table from the goal to the start. Another challenge comes from this environment being slippery. That means we have a 33% chance of not going in the intended direction. This could lead the model to choose a direction that isn't necessarily the best but will reduce the chances of us slipping into the whole. We will see how these two concerns play out when we look at our policies.

Secondly, the challenges of the BJ environment. Unlike the FL environment, we do not know everything. We knew where the holes were, and we knew everything about the state we were in. In BJ, there is a level of uncertainty and unknown we have to account for. For example, we don't know the state we will transition to when we chose an action. This is due to the randomness of the card that will be flipped over and the card the dealer has hidden. This makes our decision structure more of a conditional state, if I have this AND the dealer has this then I Call/Hit. This is how our policy will be constructed as well. This would cause a model like Q-Learning to be more likely to be more cautious and less advantageous than policy and value iteration. We will see how this plays out when we inspect our policies.

Model Explanations

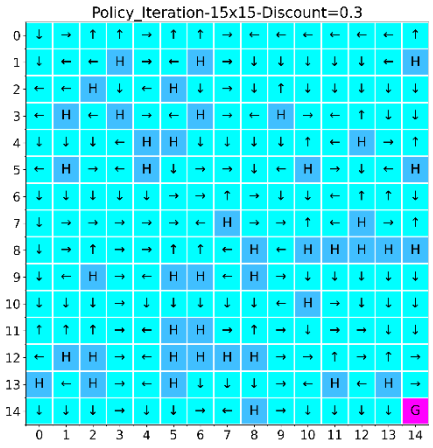
Before we continue, we need to talk about these models: PI, VI, and QL. To start off, we need to explain the Bellman equation. This equation $[R(s) + [\gamma * \max (\Sigma T(s,a,s') * U(s'))]]$ is what drives our first two equations. Essentially, we take the reward of the state and add it to a discounted (γ) max of our transition probability times the utility of the next state, then we use this value to update the utility of our current state. Even though we are starting with arbitrary values for utility, we are adding truth [R(s)] to the discounted expected utility of the next state. So, over time we are adding truth to our estimates over n number of iterations, then over time it will converge. This practice of updating the utilities in each iteration is more commonly known as value iteration, VI. There is another algorithm similar to PI, but instead of caring about utilities, we care about the actual policy. To start, we make random guesses for our policy. We then evaluate that policy by calculating the utility with the equation $R(s) + [\gamma * \max (\Sigma T(s,\pi_t,s') * U(s'))]]$. Notice how instead of an action in our transition function it is the policy, because we already know what action we will take due to the arbitrary policy we set up at the start. Once we have this calculated we pick the actions that give us the most reward and update

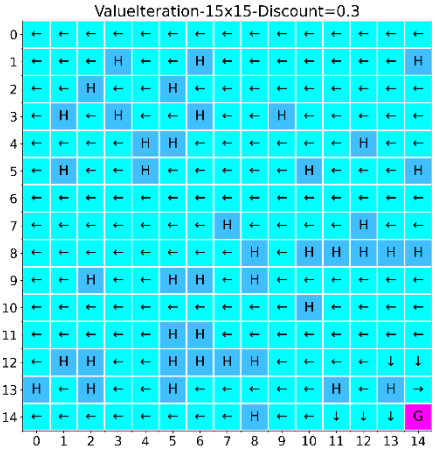
the policy to contain those actions and in the next iteration the actions that are chosen will try to get us to those actions. This is more commonly known as policy iteration, PI. These two are model based learners as they learn the optimal behavior by learning the model of the environment through actions and observing the rewards of the current state and the one that follows.

The last algorithm we need to explore is Q-Learning. QL is a non-model based learner in that it learns the policy by trial and error. It doesn't use the transition probability matrix which is most commonly referred to as the 'model' of the environment. Instead, it goes through trial and error and records the actions that give it the best reward. The equation for QL is Q(s,a) = R(S) + γ*MAX(Q(s',a)) or in other words, we estimate the utility of the current state by adding the reward of our current state to the discounted estimated max utility of our future states. We then take this value and update our Q-matrix at the appropriate s,a location with that utility and then we rinse and repeat for a large number of iterations until we converge. It is important to note 'a large number of iterations' as we must approach infinity in our iterations in order for a QL algorithm to actually converge.

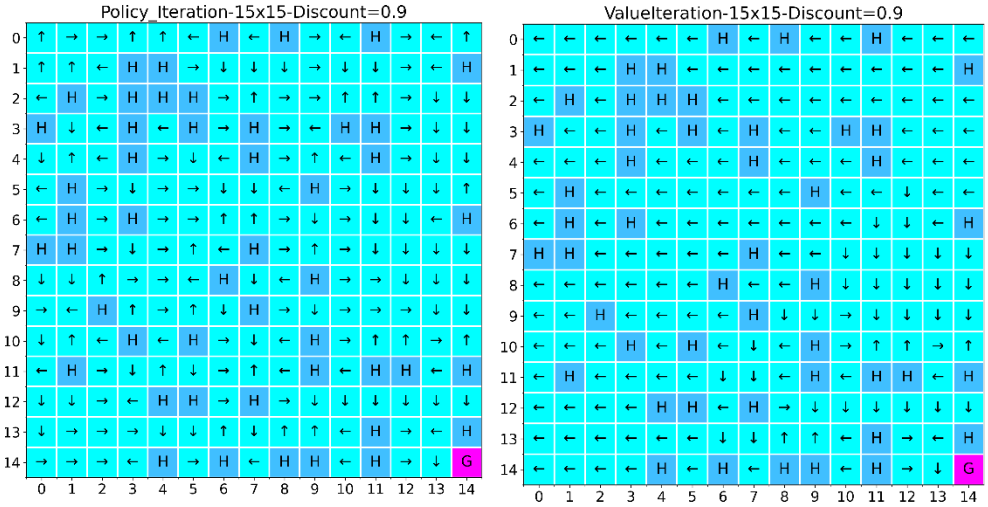
Frozen Lake

Now that we have explored and set up our algorithms, environments and procedures we need to start analyzing the results. We will begin with FL. As mentioned above, we will begin looking at PI and VI and we will start by looking at a 15x15 state with a discount of 0.3. To the right we have the resulting policy from this scenario. Remember, we have 4 actions: left, right up, and down. Each action corresponding to 0,3,1,2 respectively and the default being 0 or left. As you can see, policy iteration does a better job at moving away from the default value, but not necessarily the best policy as our discount is rather low and so we aren't learning much from our future states. We know this isn't the optimal policy because when you look and (10-13,14) we see they are pointing down when they should be pointing towards the goal, at bare minimum at least (12,14). Value iteration struggles with this map as a whole due to the utility being mainly 0 and we aren't able to update the utility of each square as the information retained from the bottom right to the

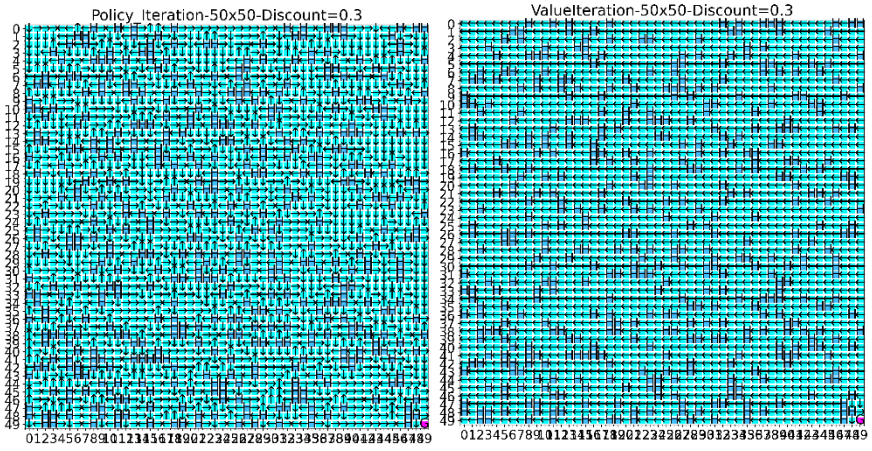




top left is very small and eventually approaches 0. But we do see some promise as we get closer to the goal. What would happen if we were to increase our discount to .9?

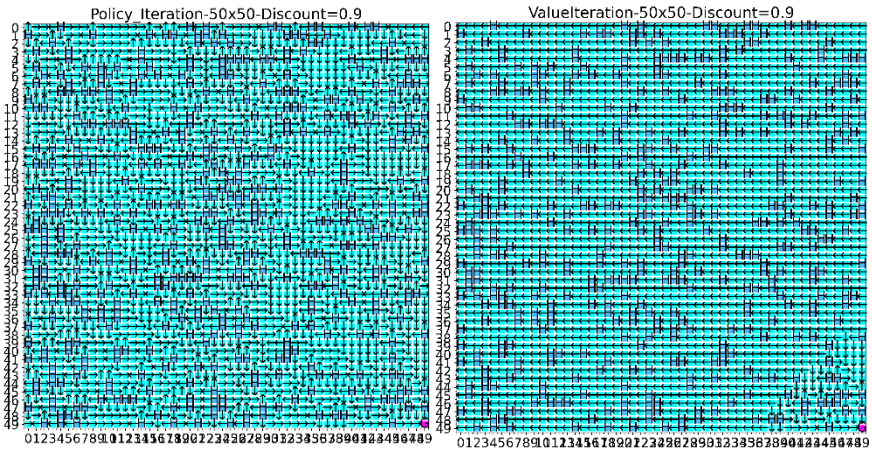


To the left we see the results of a gamma = .9, please note that due to the randomness of establishing the environment I wasn't able to maintain the same location of the holes. But the probability of the hole being frozen has remained the same and should have around the same number of holes as the first problem. Note how in PI the decisions seem more deliberate. We are able to see the flow from the top left to the bottom right. There is a key behavior in Pi that I would like to address. Notice how, for example (9,5) all arrows are pointing away from the hole. This is a key feature because we have a probability of slipping either to the left or the right of our decision. If any of those arrows were pointing in a different direction, there would be a chance of slipping into the hole and getting a negative reward. As we are focused on the policy and not the utility of each state, it makes sense that the optimal policy would be making choices that best remove the chances of falling into a hole. Also, note how in VI we are able to propagate the utility of each square future away from the goal that resulted in a better decision. If you were to draw a diagonal line from (6,14) to (3,14) that is how much of the space, we were able to update with meaningful utilities that didn't result in a default value of 0 as our action. Notice the behavior of moving away from the holes isn't present as we are only focused on the utility of each square (ex (6,10) in both PI and VI). Now that we have identified some key behaviors between low and high discounts, let's examine higher state space of 50x50 with their varying discount sizes.

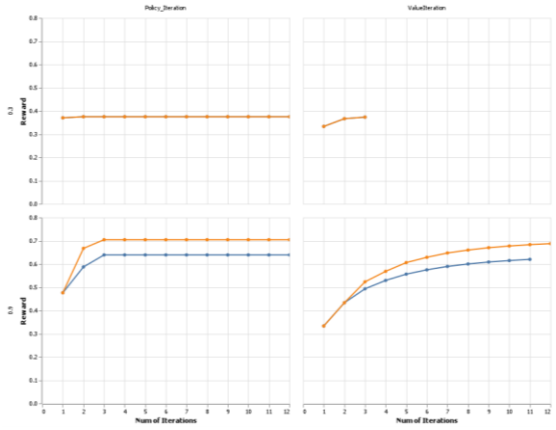


To the right you will find the resulting policies of this state space change. It isn't surprising to see a big difference between the policy of these two samples spaces. We see PI doing a better job of moving away

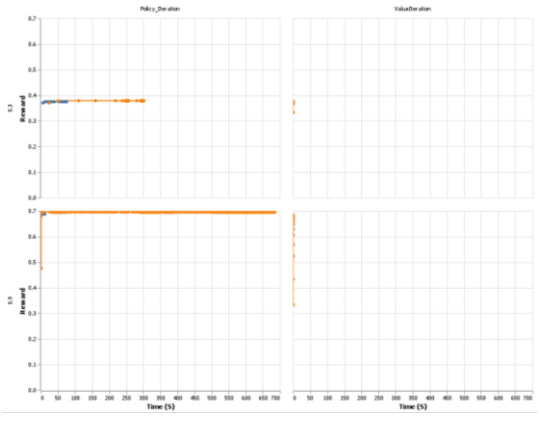
from the default value and trying to get to the end, meanwhile VI is still encountering the vanishing utility as it gets farther and farther away from the goal. It is a little hard to see from this size of a graph, but we also see the same moving away from the holes for

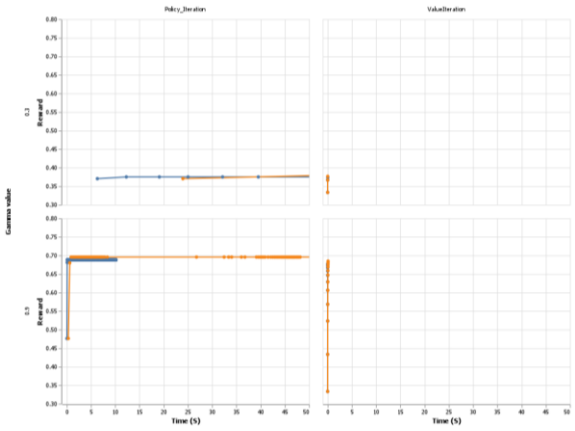


PI as we did in the smaller chart. It is safe to say that we didn't come across any behavioral difference in these policies, but instead we came across performance differences. For this, we will have to look at some convergence charts, specifically iterations plotted against reward.



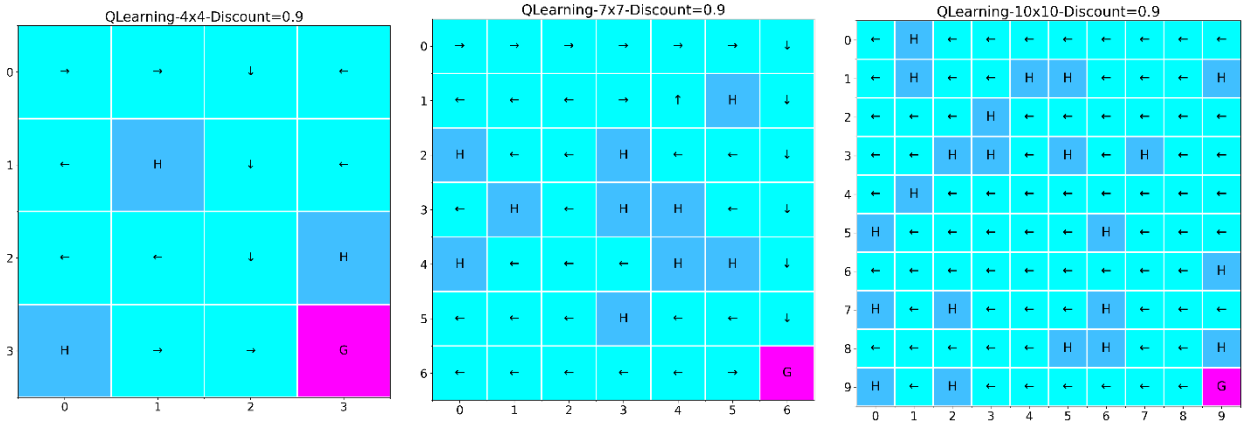
To the left we have a convergence plot with iterations on the X and reward on the Y. The columns representing PI and VI respectively and .3 and .9 for the gamma values on the rows and blue representing our 15x15 board and orange the 50x50 board. I stopped the chart at 12 iterations as the .9 gamma continued on till around 50 iterations but there was no change in the line, this is the point at which the algorithms plateaued. For a gamma of .3 there wasn't a big difference in performance due to us valuing our current state's reward (0) more than the future reward and so our overall reward is very low. But, with gamma being .9 we value the future more than the present and so we are able to reach a better reward the larger the state size, although it takes longer to converge, but how much longer in terms of time? Below is a graph that shows the time. Note how even though PI had a better policy, the time it took to achieve that, especially as the





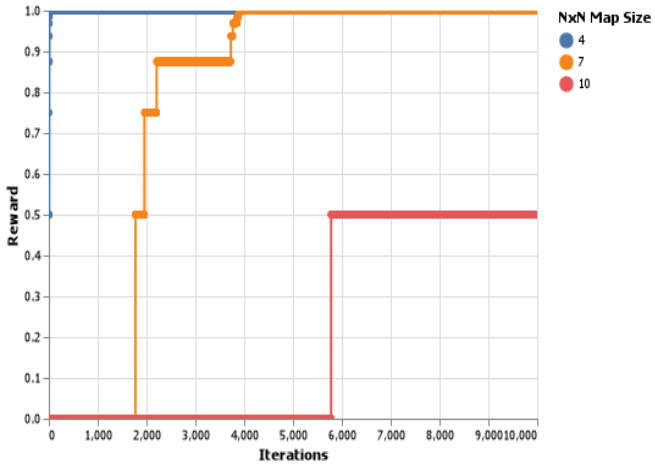
sample size got larger and the gamma got higher went up by a lot. The longest time was around 675 seconds for PI where VI had a max of around 2 seconds. So, it is a very intensive game of tit-for-tat between time, sample size, and which algorithm to use.

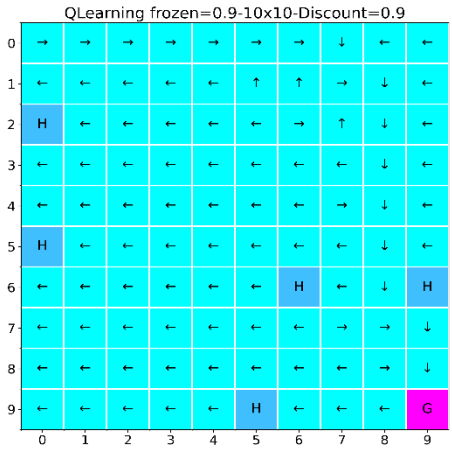
Now, as we examine QL one of our possible pitfalls comes into play. QL relies on feedback at every step of the decision-making process, but due to the majority of the of the states being 0



and QL falling into a hole before it can find the goal it doesn't perform well on a high state count scenario. To demonstrate, look at the policies below.

These charts have the same gamma of .9, yet the behaviors are so different. In the smaller size of a 4x4 the algorithm can find the goal by (RRDDDR) and then all other values are inconsequential because QL only cares about the best possible route in this case, with (1-2,3) pointing us directly at the goal as the future state of (2,3) is a reward of +1 and then we are able to propagate that reward back to (1,3). When we look at the 7x7 field QL again finds the best route and forgets about the rest (RRRRRRDDDDD) with the arrows around the immediate vicinity of the goal pointing towards it. We can overcome this by changing the reward structure of the game where every step that gets QL closer to the model receives some small reward, but I wasn't able to figure out how to write a wrapper around the engine to allow this to happen. But our convergence plots show interesting results, above. Notice how in this convergence plot, QL can reach the best reward rather quickly in terms of iterations, but when

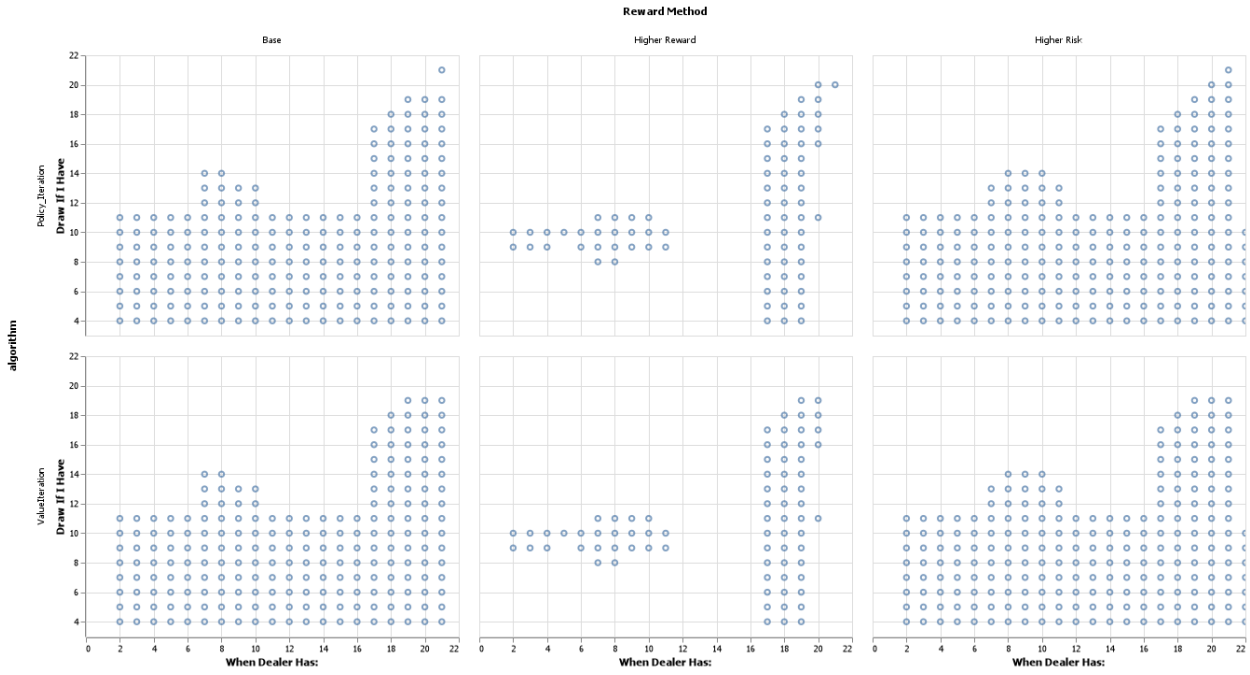




there are more holes on the map it becomes harder for QL to reach the goal and thus the reward returned is 0. Another way to fix this is to increase the chance of a tile being frozen from .8 to .9, this will mean more tiles are frozen and QL has less chances to fall into the holes and get stuck. This tells us that QL is heavily reliant upon receiving positive feedback a.k.a. a reward from every single iteration or it will fail to create an optimal policy.

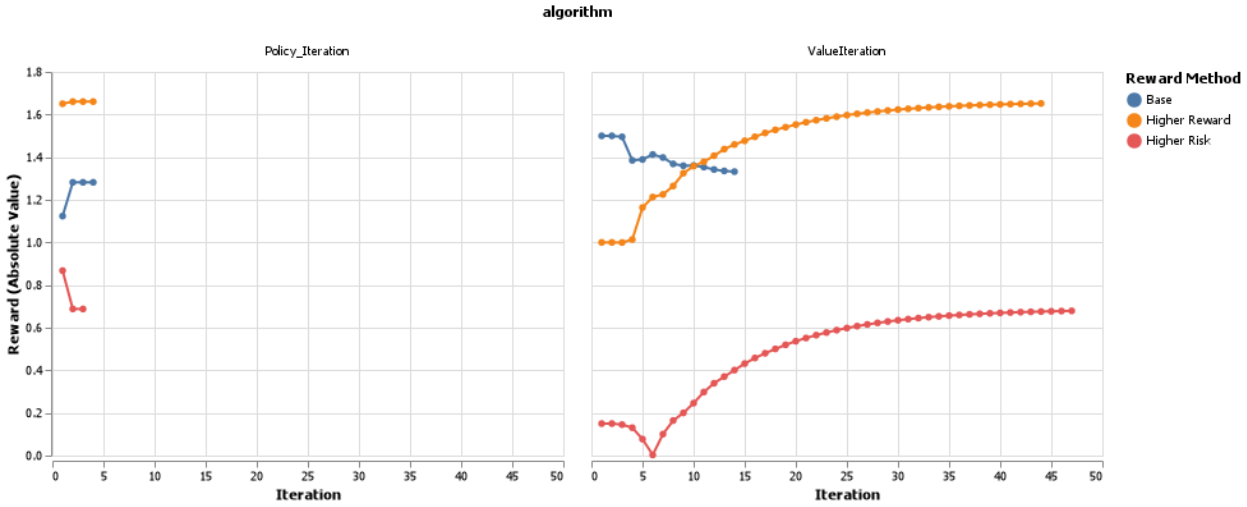
Blackjack

From the last experiment we learned a higher gamma means we value the future results of future states than the reward of our current state. Thinking about the BJ environment this is exactly what we want because we don't have a reward in our current state until the end of the round. We have to think 1-2 steps ahead in order to win the game. So, with a gamma set we can look at the resulting policy for VI and PI below. On the X axis we have the number of points the dealer has and on the Y axis we have the number of points we have. Take for example 10,11 on the bottom left graph, because there is a point there that means we should draw, or when the dealer has 10 points and I have 11 the best course of action is to draw a card. But at 10,16 or when the dealer has 10 points and I have 16 points I should call because I am currently winning, and I don't want to go over 21 points. For the columns we have the reward method, and the rows are PI on the top and VI on the bottom.

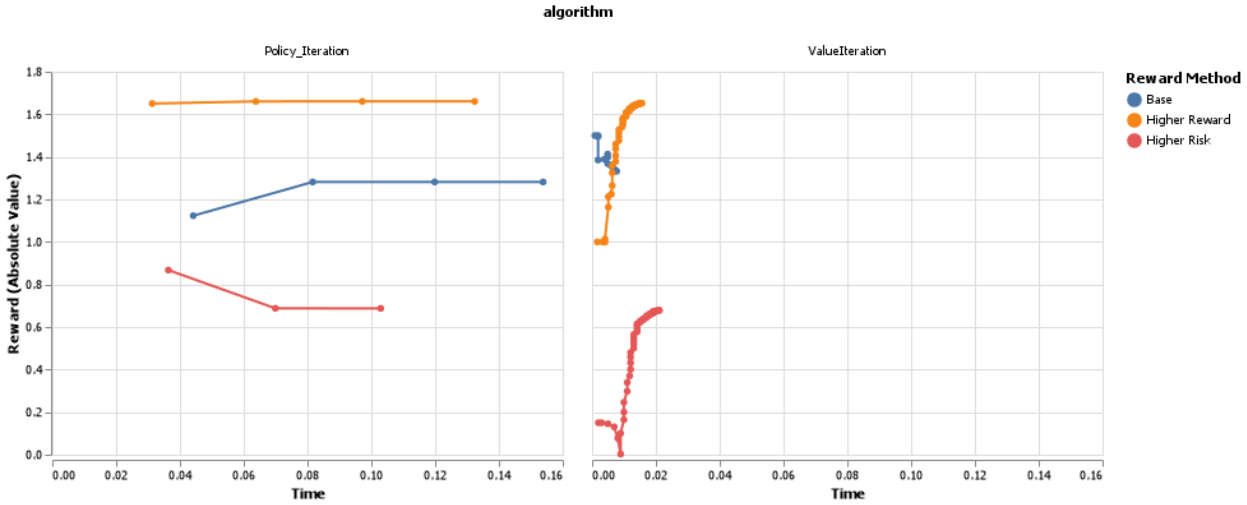


There really aren't any meaningful differences between the two policies, but there is a big difference between the reward structures. Something I thought was interesting though is when there is a higher risk the model chooses to draw when we are below 10 points, but the dealer has 22 points, meaning he has already lost. The model wants to avoid losing so badly that even though it has already won it will still try to get closer to 21. There isn't a big difference between

the base and a high risk as the model is already trying to win, but there is a huge difference between the base and high reward. It seems the model has begun to believe that it has already won in circumstances and will only decide to call when it is most likely a loss, meaning the dealer is close to 21, or when they have around the same number of points. How interesting that a higher reward would teach the model to be more cautious and be ok with losing some rounds. Now, let's look at some convergence plots.

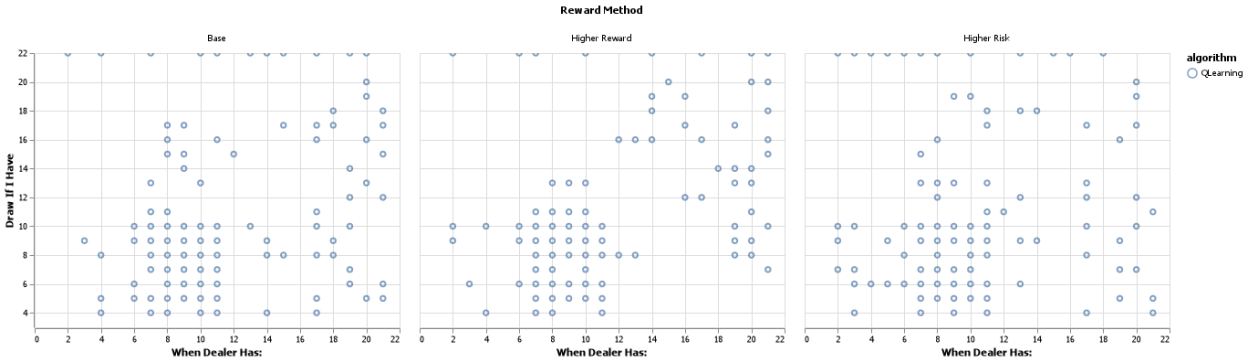


Above you will see how quickly PI was able to converge compared to VI. I believe this is happening because policy iteration starts with a randomly selected policy which will most likely only take a couple of tweaks to get correct in such a small action space of call or draw. Whereas VI is concerned about the utility of each state and will continue to update the utility as long as it is changing significantly. Due to the varying number of states and some ambiguity within this environment, it makes sense that the utility of each value would be changing significantly within each iteration. As for our time convergence below, it is rather surprising to see VI take



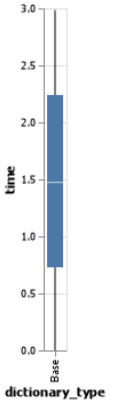
less time to compute than PI as VI is more computationally heavier. Also, that VI was able to maintain a high of less than .01 whereas PI approaches .16 for the base case. This might be due to the idea that even though it is taking more iterations, VI only has 2 action spaces to search

and so it is only searching across 2 spaces where as PI is searching across the entire policy. Furthermore, each iteration of PI is increase roughly by .03 each time meaning it is getting more computationally expensive the longer it things about it.



Finally, we look at QL. Now this graph has no consistency to it at all, maybe you could argue between 6 and 11 on the x axis and 4 and 10 on the y axis. It took me a long time to think through why QL couldn't converge to an optimal policy even over 30k iterations which is 3 times as much when compared to the FL problem from earlier. But you have to look at the consistency of the states in order for the answer to come to light.

Let's take a step back and look at FL. How many states were there? 4x4, 7x7, 10x10, and even the model could handle a 15x15 space or 225 distinct states. The key idea is the model was able to see ALL those EXACT SAME STATES every single iteration, thus learning to generalize a policy from consistent data. For BJ, say there is only myself and the dealer with 13 cards in the deck. There are 52 cards and we don't care about suits so we say 52!/4! Different possible ways to organize the cards, but we don't care about all the cards, just the first 4 (2 for me and 2 for the dealer) so this turns into 52!/4!(52-4)! = 52!/4!48! = 270,725 different possible combinations and we haven't even gotten to the first draw yet! QL, a model free algorithm, relies on being able to experience the same transitions over and over so it can learn what choice to make when it is in those situations. For our 30,000 iterations, assuming they are all distinct and none of them were the same, that means we only encountered 11% of the possible combinations and so our model has no way to generalize or form a policy because it has never seen any state more than 1 maybe 2 times if it is lucky. It does its best by finding the expected value (the middle portion mentioned earlier) but anything outside of that is nothing more than a random guess or the results of playing it one time. Say we wanted to make sure the model saw each state twice, and we had the power to make sure it was sample without replacement until there was nothing left, we would need to do 270,725*2 or 541,450 iterations! Keep in mind this doesn't take into account if the dealer and I were to have reversed hands, because that is a totally different state as well.



To see if this is possible let's plot the distribution of time per iteration in a box plot. As you can see the mean time is .8, 1.5, and 2.3 seconds for the 25th, 50th, and 75th percentiles. When we multiply these values by the number of iterations, we

explored above we get 433,160 & 812,175 & 1,245,335 seconds or 7,219.3 & 13,536.25 & 20,755.58 minutes or 120.32 & 225.6 & 345.92 hours respectively. I don't really feel like running that kind of computation time on my poor laptop. Keep in mind this is the number of combinations PRIOR to starting the game. This is just to get the cards dealt to the players. I know the model was trained on time including the next few draws, but if we included the next 2 draws in the game that would make the number of iterations needed go up really fast and already, we are at a little over 5 days assuming none of the iterations take more than .8 seconds. Which I guess is why they say to never gamble because the odds are never in your favor, and you will never see the same state twice.

Conclusion

After exploring these different models across different environments, I have come to appreciate the complexity of reinforced learning. MDPs are not something to be trifled with and it takes a lot of synthesizing and time spent with these models to truly understand what is going underneath the hood. We have learned QL needs a consistent state space in order to build a decent policy, and the number of positive and negative rewards in your space need to be balanced otherwise QL will never find an optimal policy. VI is a very computationally expensive algorithm which can take many more iterations than PI. But on the flip side to that, PI in simple action spaces will be out done by VI in terms of computation speed. VI has a hard time assigning meaningful utility to states that are furthest away from the result because it is hard to know if that action truly the cause of you was losing (think of your first move vs the last 3 moves in a chess game).

Attributions

MDPToolBox-Hiive GitHub: https://github.com/hiive/hiivemdptoolbox

GYM: Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., & Zaremba, W. (2016). Openai gym. *ArXiv Preprint ArXiv:1606.01540*.