Machine Learning Project 4

Forest: Introduction

For my first Reinforcement Learning problem I picked the non-grid forest world. The default world allows us to cut (reward 1) and waiting (reward 0) but in the last/oldest state the reward to cut is 2 but wait is 4. Additionally, there is a 10% chance the forest will burn which reverts it back into its youngest state with no reward for that time. I focused on 2 sizes a small at 150 and large at 900. The best Policy in a simplistic sense then is to always cut (reward 1) but if we find ourselves in the last state to wait instead of cut (reward 4 instead of 2). Since the probability of fire is 10% shouldn't we wait if we are within 10% of the end? So we also have to weigh the reward of wait against the cumulative reward of cutting for that last 10%. In our case this would exceed the wait reward of 4. But even in the penultimate state an argument can be made to cut instead of wait. Cutting would allow us to gain a reward of 3 (1 for penultimate and 2 for last) vs a reward of 4 for waiting and no fire. Is it worth the wait? Additionally, even cutting can lose reward by the 10% chance of fire and be reduced to 1, 2 or even 0 if fire occurs twice. This is more of an open-ended domain question. The Reinforcement Learner can be made to reflect how highly we value future rewards by tuning Gamma or discounted rewards. Alternatively, we can shape our world to have a very high reward for waiting in the last state which makes it worth it for the Learner to accept a higher risk tolerance for a higher reward. I kept the reward the same and tuned Discount in the graph below.

VI/PI: Small

I first focused what makes a good Epsilon for VI. I scaled them logarithmically and found that they overlap each other. This makes sense as Epsilon is a measure of how far we should drive down the error. A smaller Epsilon will follow the path of a larger Epsilon and continue further by driving the error down more and getting more reward. However, there is a trade-off between accuracy and time. A smaller Epsilon means less error and more reward but also more time. So, we look for the greatest gains proportionally and .1 seems to crest that curve and stop at the point of diminishing returns. This is also very fast and the differences in time basically imperceptible at these Epsilon and Policy sizes.

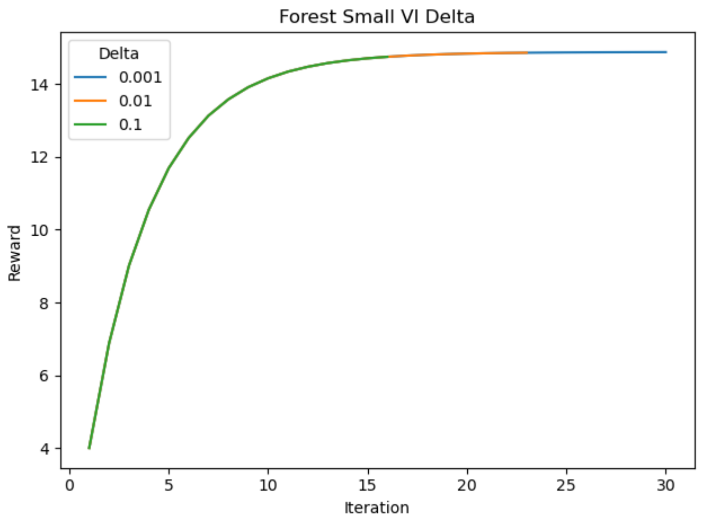

Figure 1: Forest Small VI Epsilon

Plotting for Discount shows us this trend of valuing later reward. As we see Discount affects how much we value future rewards. If we have a high Discount this will value the future rewards more and a low

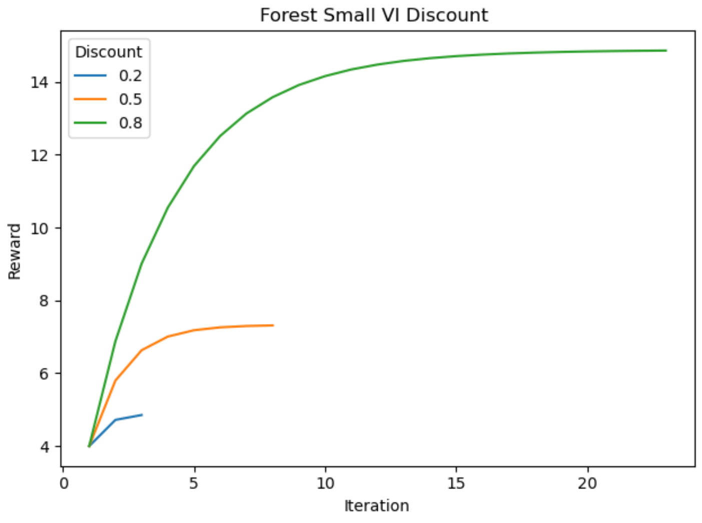

Figure 2: Forest Small VI Discount

discount less. I would argue that the rewards in this graph are all equal, it's just that Discount can inflate or deflate them and allow the Learner to have different value systems. Do we want the Learner to focus on short or long term rewards? Note that a higher Gamma/Discount leads to more iterations. This is because it propagates values further and needs more cycles to propagate and achieve Epsilon convergence.

For PI I set the Epsilon to .00001 and iterated through policies to find the best one. It converges pretty fast also and mimics VI

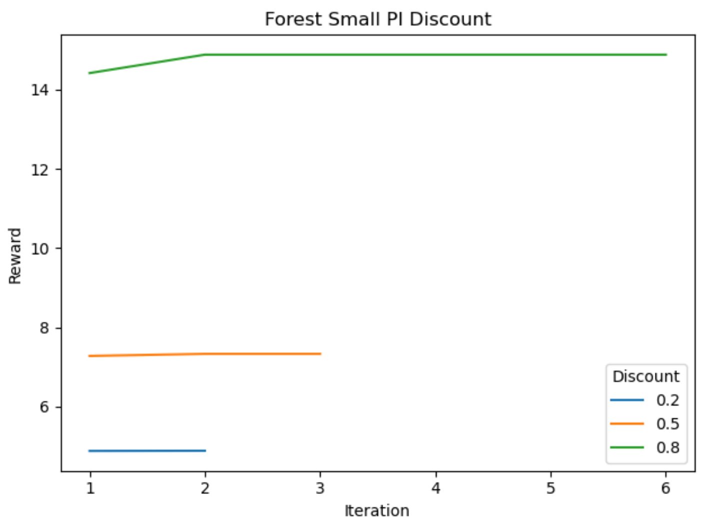

Figure 3: Forest Small PI Discount

with it's Discount returning the same Reward. However, it seems to start there and only .8 shows actual improvement. I initialized the policy to have an 50% of 1s and 0s at random to create a random policy to start from. However, having all the information in the form of the Transition and Reward matrices seems to allow small PI and VI to converge extremely fast, both in iterations and time. PI converges in 6 iterations and VI in 25 both very small in comparison to Q-Learning.

Q-Learning: Small

For Q-Learning convergence is guaranteed given (some conditions that we satisfy and) an infinite of time (Mitchell, p. 377), (Sutton Barto, p. 131). Unfortunately we don't have that so I defined convergence by what a reasonable amount of time/iterations would be to get us the best answer per the problem size.

The first hyper-parameter I tuned was Alpha. Alpha is the learning rate of the model. As we see .3 here is the best alpha with .9

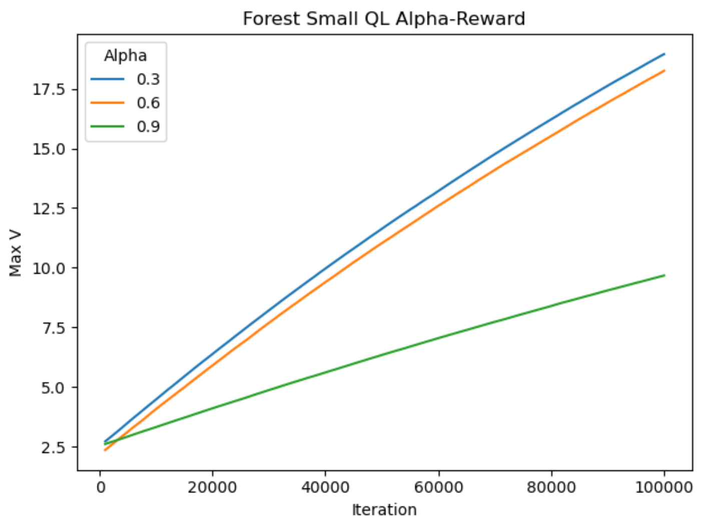

Figure 4: Forest Small QL Alpha-Reward

the worst. This illustrates that if alpha is to high the learner will keep learning and won't internalize what it already learned. It will keep valuing new information highly which doesn't allow it to converge. Note that .9 exceeds .6 right at the beginning. This is because it learns faster and will value that new information more highly. This is an asset at the beginning but a hindrance later. A good analogy is throwing a ball. A lighter ball (.9) will have more acceleration but less momentum and can veer off course. A heavier ball (.3) will have a slower acceleration but more momentum which keeps it on track. To low of an alpha would hurt also as we wouldn't be able to learn effectively. The learner would need many more iterations to converge. So to high an alpha and we never converge and to low will take to long to converge.

Another important hyper-parameter to tune is Epsilon which in the context of Q-Learning means how often will we take a random action. Though it looks similar to Figure 4 it has key differences that highlight the affect of Epsilon. In the beginning of the graph all the lines seemingly have dif-

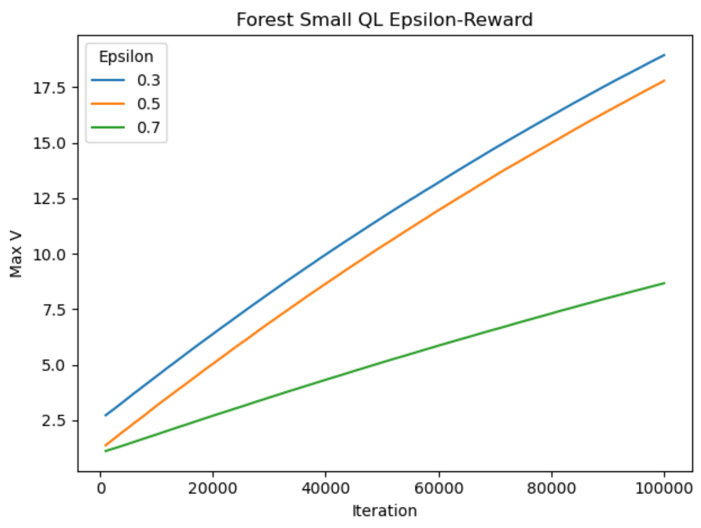

Figure 5: Forest Small QL Epsilon-Reward

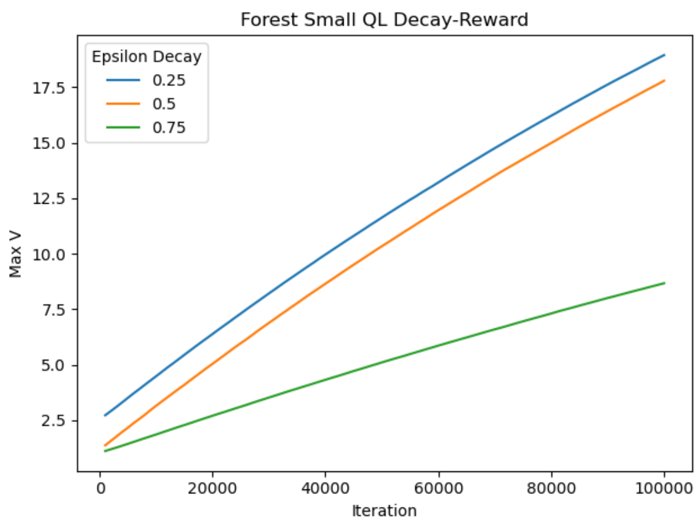

Figure 6: Forest Small QL Decay-Reward

ferent starting points as opposed to Figure 4 where they crossed in the beginning. This because a lower Epsilon is less likely to explore and will choose the known and better action more frequently. So even if we set Alpha to .3 which we determined in the graph above to be best; A high Epsilon will not allow us to realize that policy as we take random actions to frequently for that policy to be of use.

To counter-act the continuing use of random actions is another hyper-parameter called Epsilon Decay which reduces or decays Epsilon by a certain percentage. This allows us to Explore more in the beginning and eventually settle or Exploit what we learned into a policy and take fewer random actions. We chose .3 as the best Epsilon and the Decay rate here is .25. So we start with a low random action rate but have a slow random action decay rate. This makes sense and seems to be a trade-off. To have a high random action rate means that we take random actions pretty frequently which leads to high exploration. To counter-act that we would need a high decay rate so we don't take those random actions indefinitely and instead settle down (similar to Simulated Annealing) into the optimal policy while not taking random actions. In our case we start with a low Epsilon/Exploration rate and we decay it slowly so that we can keep exploring. At the last stage we will have the lowest random action rate and will be Exploiting the most.

VI/PI: Large

I set large to 900 states for forest which is 6X the small number of states. The most startling difference here is in the PI Discount policy. I used an initial policy of random 1s and 0s just like in the small example. However, here we see the gain by the policy iteration. It still converges rather quickly and this is still due to having all the domain knowledge up front. As with value iteration a larger Discount/Gamma has to propagate its values outward in order to reduce Ep-

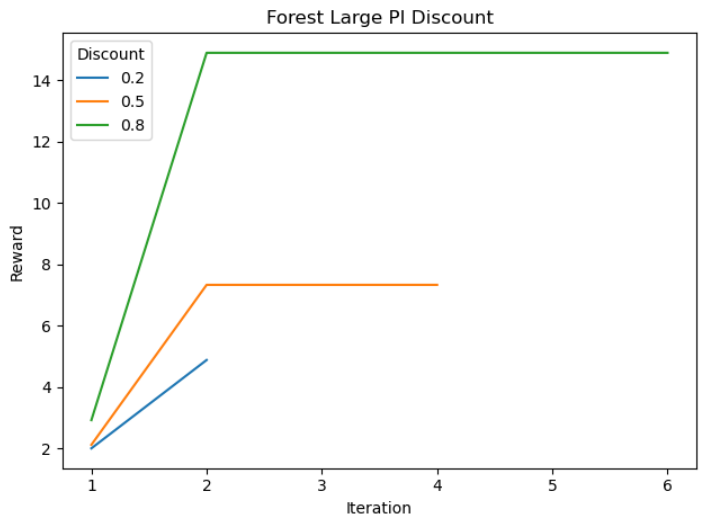

Figure 7: Forest Large PI Discount

silon to the correct degree in the policy via iteration. The higher the Discount the further the value will be propagated outwards which leads to more iterations.

One difference in the VI policy is the time. Though the curves look basically the same (again if we have all the knowledge we can do all the calculations pretty similarly) it takes longer in bigger state spaces. As we

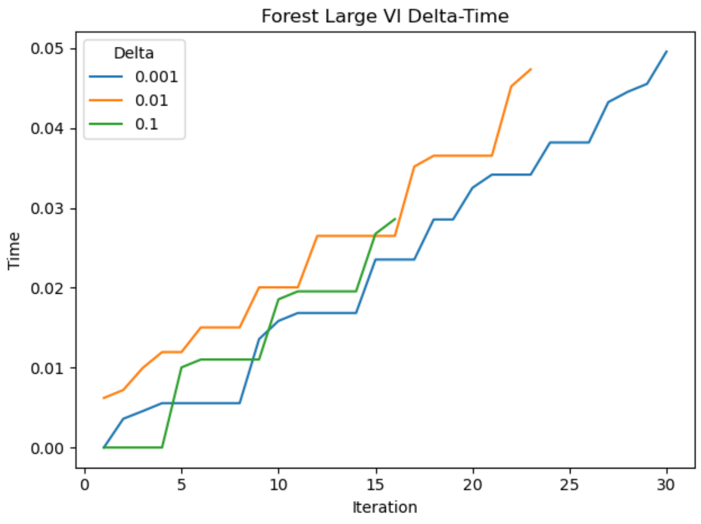

Figure 8: Forest Large Delta-Time

see a smaller Epsilon will lead to more iterations as we need to drive the error down further for convergence which needs more time. It is interesting to see that though each Epsilon is 10X the orders of magnitude smaller it doesn't take 10X the time. This is because there is a lot of error before hand that they all have to work through. As they approach Epsilon convergence the time will be less in between the different values of Epsilon. This is kind of like a limit of diminishing returns of Error vs Time. Additionally, increasing the state size did not increase the number of iterations needed but did increase the time. I believe this is because there are more calculations needed for a bigger state space per iteration. This would lead to the same number of iterations but more time.

Q-Learning: Large

We started by using 6X the number of iterations as the problem size was 6x bigger. Interestingly, proportionally it seemed to converge faster and we can see the curve cresting in the graphs. This graph reinforces

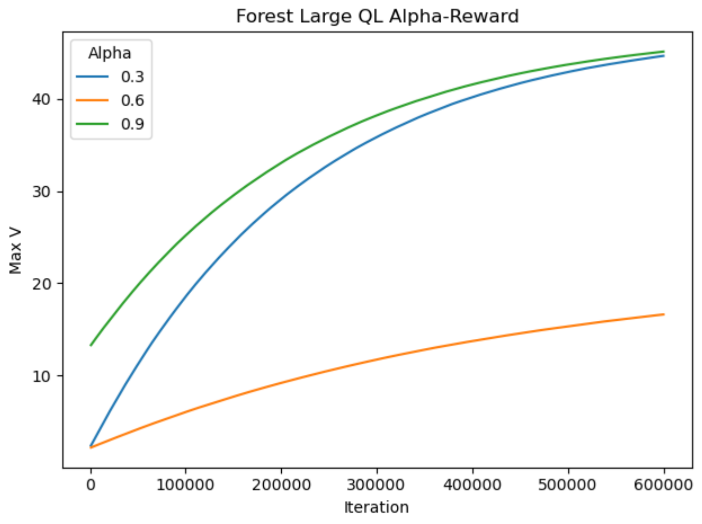

Figure 9: Forest Large QL Alpha-Reward

what we said earlier that a lower alpha converges slower in the beginning than a higher alpha. But why does the middle one perform so badly? Perhaps the probability of fire where it doesn't get reward despite the correct action is throwing it off. A lower alpha shrugs it off as anomaly and a higher alpha just corrects itself next time around. In the middle however leads to indecision.

Additionally, we have the time component of the Learner. While the Epsilon and Decay time grew linearly with the number of iterations Alpha was different. As we saw

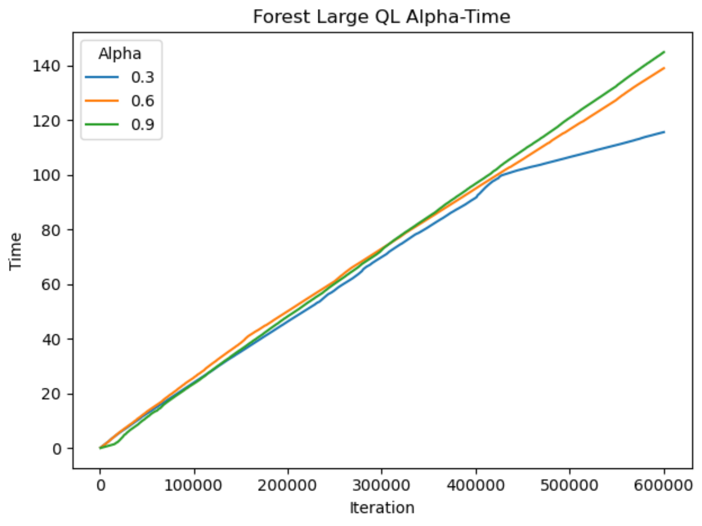

Figure 10: Forest Large QL Alpha-Time

in Figure 9 .6 didn't converge so it makes sense that it takes longer as learning will take longer. But if both .9 and .3 have converged why does .3 take a shorter amount of time? Perhaps the answer is a combination of what we just said and what we noted earlier. As mentioned .6 is still learning and learning takes time. At convergence though we should just be returning the policy which is pretty fast. But we also mentioned that .9 is still "learning" as its Alpha remains constant. So perhaps the Utility may have stabilized and the learner has a policy similar to .3; but it has to continuously "learn" that policy which takes time.

Forest: Conclusion
For Forest we generated optimal policies using VI, PI and Q-Learning. We saw how PI took the shortest amount of iterations, followed closely by VI, and Q-Learning required thousands if not millions iterations to converge. For VI/PI a larger state space didn't mean more iterations. It still had the same amount of iterations but the time did go up. This is because each iteration has more to do in the larger space. Additionally, we saw that VI/PI converged much faster than Q-Learning. However, they also had access to all of the worlds information; without that information they would be helpless. Q-Learning doesn't need the worlds information which makes it more versatile and usable for worlds where we don't have the Transition and Reward matrices. In trade it takes much longer as it has to explore the world on its own. Another factor that slows Q-Learning down is that we don't want it always exploring, at a certain point we want it to start using or exploiting the knowledge it gathered which trades exploring time to exploiting time. Also, Q-Learning needed more iterations for a larger state space as opposed to VI/PI which kept the number of iterations the same.

Lastly, the optimal policy by default is to cut for both large and small. It is hard to graph the optimal policy (large would be a string of digits 900 long) but easy to gain intuition and mentally visualize. As we noted the closer we get to the end the more worthwhile it is to wait. This depends on a number of factors; some world related and some not. A lower probability of fire, higher wait reward, and lower cut reward are all world factors that would contribute to waiting. A non-world or learner factor is Gamma/Discount or how much we value later rewards. If we tell the learner to value later rewards this would also push the learner to a different optimal policy and wait in more states closer to the end.

Frozen Lake: Introduction
For my 2nd world I used the Frozen Lake grid world in which you ave to cross a Frozen Lake while avoiding holes. I added some reward shaping for both small and large worlds. Each step brings a -.5 reward with the goal reward being 100. This encourages the Learner to finish as soon as possible. If there was no penalty for moving it could wander aimlessly and as long as it doesn't fall into a hole get the same reward by taking 50 steps vs 20. If it falls into a hole the iteration terminates with the current reward. Since we want it to finish with the least steps possible adding the -.5 penalizes every additional step.

VI/PI: Small
In Frozen Lake we clearly see the diminishing returns that Epsilon creates. We had mentioned that too large an Epsilon will needlessly iterate without providing a significantly better policy. As we can see .1 takes care of most of the policy updates. Adding 13 0s only extends our precision and iterations a little further. However, when we add 1 more 0 for a total of 1e-14 the iterations shoot up but the Utility doesn't change. This reinforces the point that our policy won't improve past a certain Epsilon.

We had picked 1e-13 as our Epsilon but Epsilon can be further analyzed through the lens of Discount or Gamma. Since a larger Discount will propagate values further a smaller Epsilon will be needed. As we see a smaller Discount factor converges on an optimal policy much faster than a larger

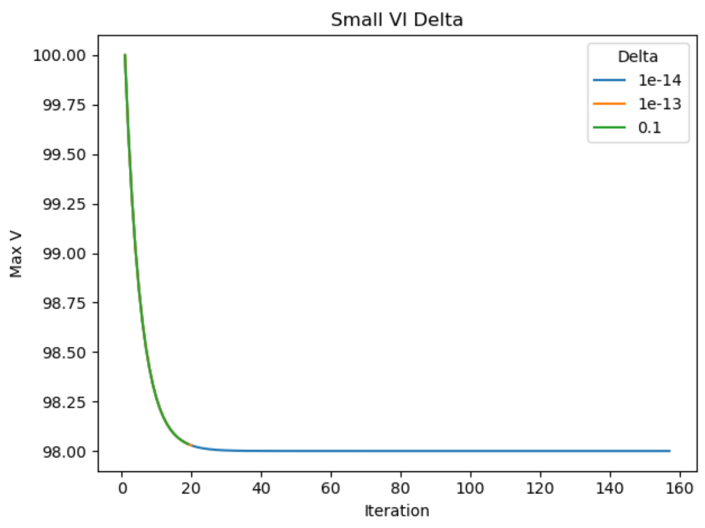

Figure 11: Small VI Delta

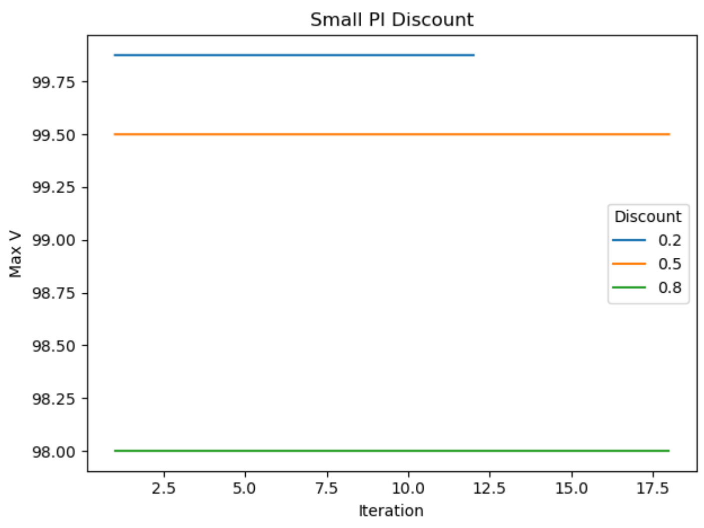

Figure 13: Small PI Discount

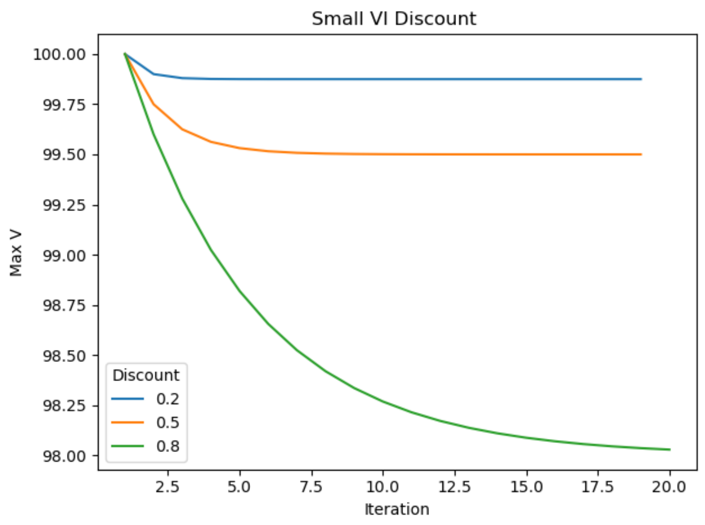

Figure 12: Small VI Discount

Discount does. Therefore, a smaller Epsilon can be used to get the optimal policy faster. The larger Discount has a smaller Max Utility as the negative rewards can affect cells that are farther than them and create greater overall negative rewards. This ends up taking longer to find the optimal policy. Additionally, VI takes very few iterations as seen here even with a very small Epsilon as we have all of the Reward and Transition matrices beforehand.

I used an initial policy of random actions and an Epsilon of .00001 which produced this graph. As we have all the information beforehand it converges even faster than VI. The reason it continues is it reduces its Epsilon in between policies. However, it produces a flat line because the Utility isn't changing in between Policies.

The optimal policy it produces is the same for VI and PI which also indicates that they are achieving the same results through different means. The directions are LEFT = 0,

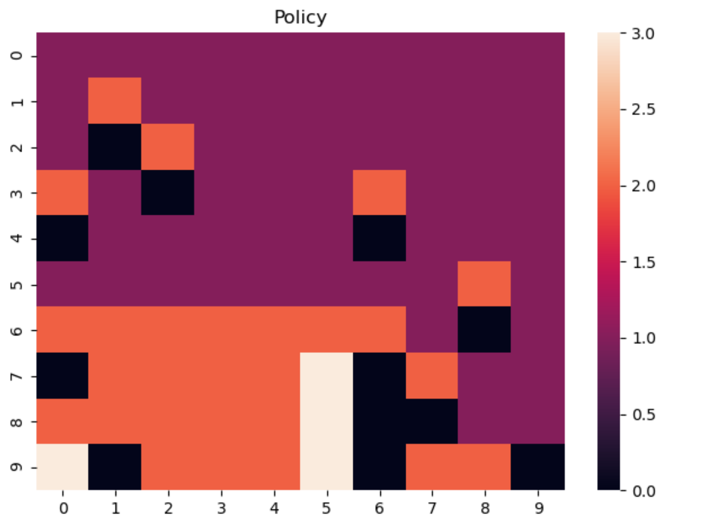

Figure 14: Small VI/PI Policy

DOWN = 1, RIGHT = 2, and UP = 3. We can intuit where the holes in the ice are where the policy goes away from the goal either by going left, up or a combination of left and up as seen in the bottom middle.

VI/PI: Large

Extreme differences appear when comparing our large and small Frozen Lake worlds.

These need explaining. Plotting again for our Epsilon convergence plot creates a nice contrasting graph. In Figure 11 .1 was

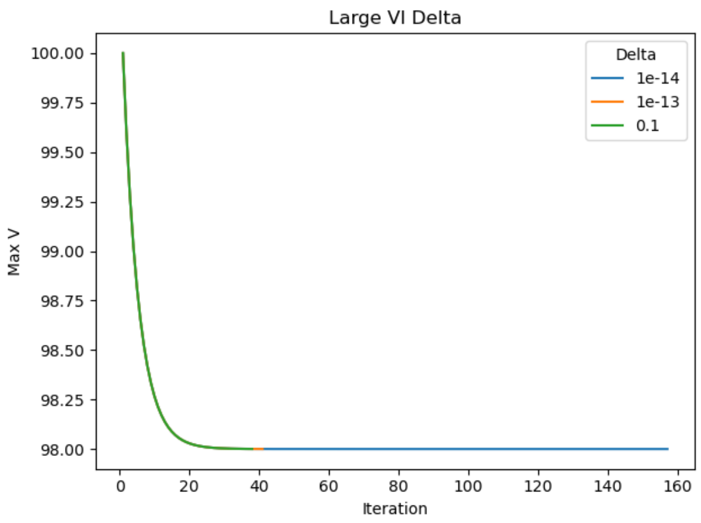

Figure 15: Large VI Delta

barely enough to round the curve of convergence vs iterations and we needed to go to 1e-13 to do so. Here though .1 is more than sufficient. What is the difference? Perhaps since there is only 1 Goal location whose value is different and all the others the same, in a larger grid world it affects the world less proportionally and it actually converges faster. Further proof to this is that in Figure 12 it seemed to iterate post convergence; whereas here the Discount ends where it should have in Figure 12. Yet we still need more iterations to hit convergence. In our Small world we hit a maximum of 20 for .8 Gamma and here we hit 40 iterations. It seems like VI/PI scale proportionally to the problem size (unlike Q-Learning as we'll see).

Additionally, PI policy does show its improvement here likely due to its larger size. Similarly to VI, PI does scale well and in the largest Discount converges at about the same iterations. However, the smaller gamma rates take longer in PI. In contrast, Forest was exactly the opposite; PI took shorter in general over VI. Perhaps the difference is the complexity in possible policies. Since PI here has 4 actions to choose from to create a policy this creates a more

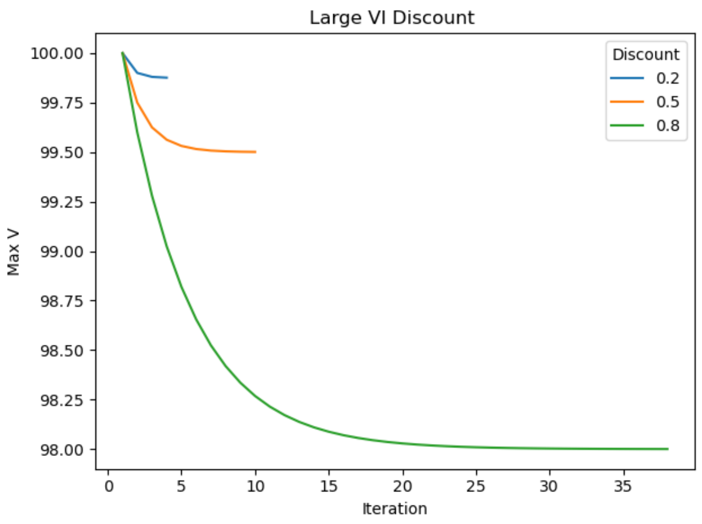

Figure 16: Large VI Discount

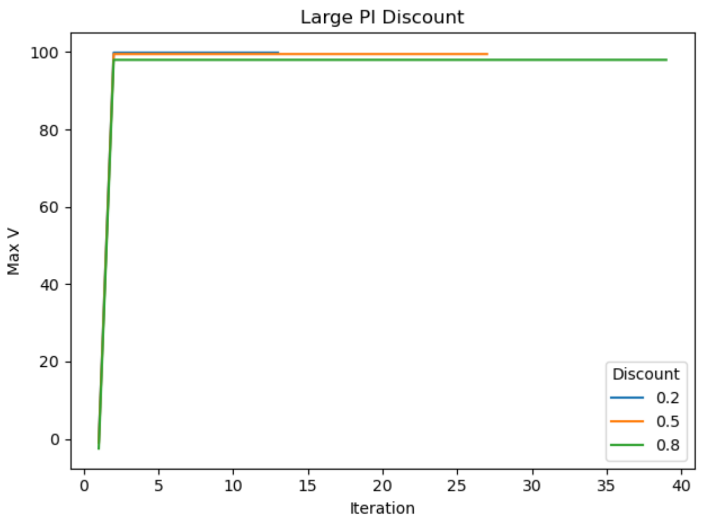

Figure 17: Large PI Discount

complex or larger policy space from which to choose and iterate over. VI just propagates the values and uses the max function to create a policy which ends up being faster with more actions to choose from. This is likely due to the max function

Q-Learning: Small

Since Q-Learning can take an extremely long time to converge we define convergence by number of iterations. As the number of iterations approach infinity any reasonable Q-Learning algorithm will end up converging (Mitchell, p. 377), (Sutton Barto, p. 131). Therefore the question becomes how can we converge in the fastest way possible. To do that we pick a reasonable number of iterations based on the space of the world and use the best values that it produces. In our small example of 10x10 which creates 100 discrete spaces we can be in we picked 200,000 iterations as sufficient evidence to pick 1 hyper-parameter over others. Of course after we have decided on our hyper-parameters we can run our Q-Learner for more iterations to assure convergence. This will hopefully reduce the number of iterations in the tuning and learning phase.

Our 1st parameter tuned is Alpha. We see

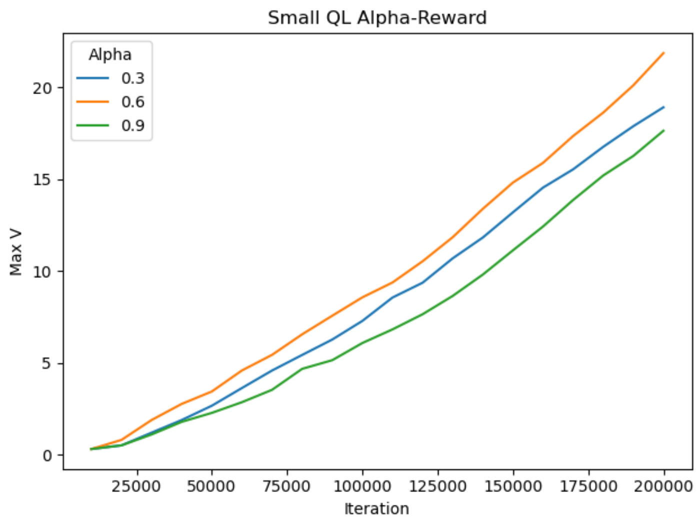

Figure 18: Small QL Alpha-Reward

how the Alpha rate of .6 is a clear winner here. This emphasizes the need to balance the speed of learning. To low or high of a rate and the Learner will converge in to long a time as it either learns to slowly (low alpha) or to fast and "forgets" what it learned previously (.9).

The next parameter tuned was Epsilon or the rate action rate. Here .7 gave us the Max Utility of the learner. I regard these results with some suspicion because perhaps the random action rate is so high is because we are still learning. So the more random actions we take, the more we learn and the more reward we get. However, if this were true our Epsilon Decay would be small; yet the best Decay rate or random action decay rate is .75 which will decay it quite fast. If

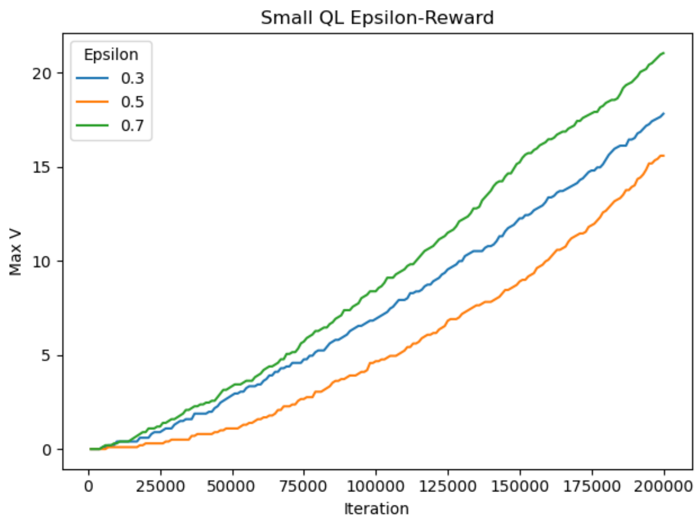

Figure 19: Small QL Epsilon-Reward

we were still learning why would a high decay rate provide the best utility? This points to .7 being a good value. As before the di-

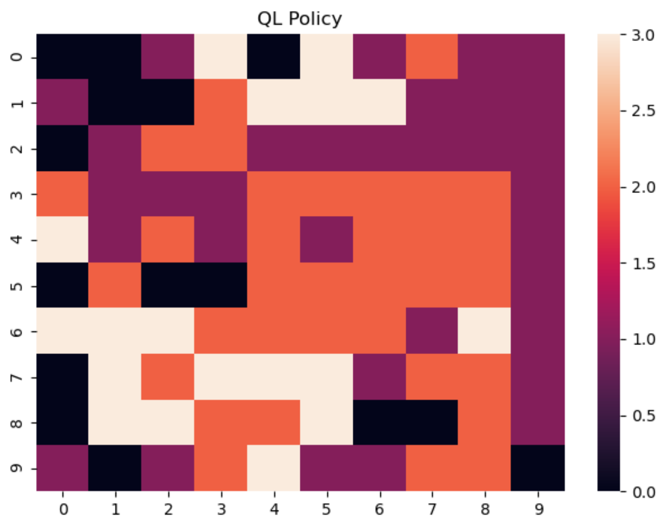

Figure 20: Small QL Policy

rections are LEFT = 0, DOWN = 1, RIGHT = 2, and UP = 3. It's not as optimal as the VI/PI policies in Figure 14 but we can see it taking steps in the right direction. We have the straight line on the side going down and a large chunk in the center going right. Honestly, there are many optimal policies here. For example, the Learner can alternate going right and down (barring holes) creating a cross-hatched graph and still be just as optimal as VI/PI. In the long run Q-Learner will probably converge on a different optimal policy than VI/PI as it concerns itself more with optimal actions. These actions are explored randomly and since there are multiple best actions to take even 2 Q-Learners can converge on different optimal policies.

Q-Learning: Large
For our large Frozen lake we upped the iterations to 400,000 and our Alpha of .6 still was the best. So we moved on to Epsilon and Epsilon Decay. So our Epsilon was best at .5 in contrast to the smalls .7. Additionally, the Epsilon Decay of .5 and .25 tied. I

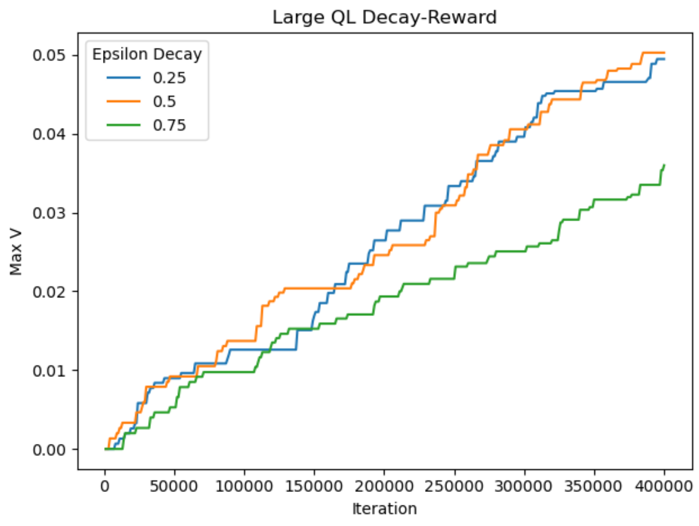

Figure 21: Large QL Decay-Reward

believe all of these factors are related. A hint that got me thinking is that our Max V is only .05 which is really small compared to 20 in our small grid world. We only doubled the size so I was expecting something around 10. Clearly, my assumptions were wrong. Thinking about it further however it makes sense. Even though we only doubled our world, the probability of getting to our goal increased exponentially. It does reach the goal occasionally which is why we have a positive reward. This is why the Learner picked .5 for Epsilon. Since we barely learnt anything random actions are nearly as good as our policy. However, since we did make it there occasionally we want it slightly less than random which is why we have a low Epsilon decay. The learner is trying to maximize what it learned by decaying the Epsilon extremely fast.

One last factor was time. As we have an

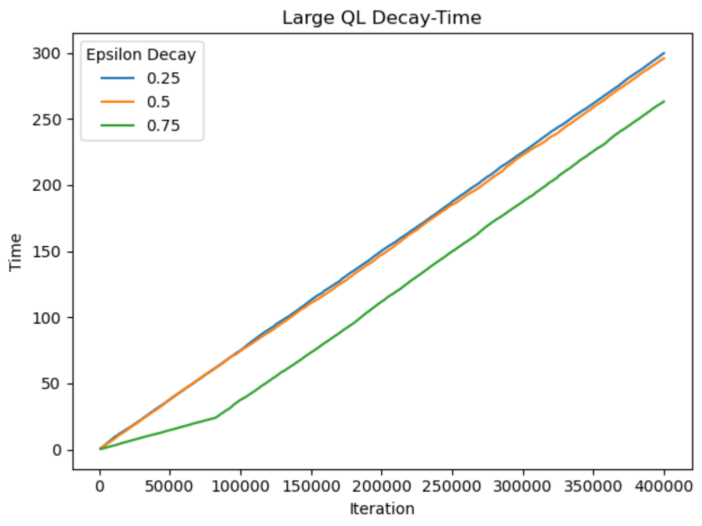

Figure 22: Large QL Decay-Time

iteration based approach for convergence we would expect basically the same time for all of the decay schedules. Why is .75 so much slower than the others? Perhaps the act of generating random numbers takes more time than following the policy. In a small way this plays into Exploration vs Exploration. The extra cost of generating random numbers slows us down and there will be more random numbers generated by .75 due to the slower decay schedule. Otherwise there is near linear growth in time. The other graphs of Time vs Iterations for Alpha and Epsilon also grow linearly.

Frozen Lake: Conclusion
Here we saw VI/PI converge in about the same amount of iterations with PI sometimes taking a bit less. They converged to the same optimal policy as opposed to Q-Learning which took a very long time to even remotely converge for the small problem size. For Large Q-Learning didn't converge at all. With 400 states and 4 actions the space was to big for it to explore and needed orders of magnitude more iterations than the Small space. Even though the large space is just double the small, the ways to get to the goal space (or exploring any space) increase exponentially which lead to extremely poor results despite us doubling

the number of iterations which worked for small. We saw small start to converge and noted that there are many optimal policies for this MDP.

References

Mitchell, T. M. (2013). Machine Learning. McGraw Hill Education.

Sutton, R. S., Barto, A. G. (2018). Reinforcement Learning: An Introduction. Cambridge: The MIT Press.