Reinforcement Learning

Sagnik Bhadra

Georgia Institute of Technology
sbhadra8@gatech.edu

November 28, 2021

I. Abstract

There are three reinforcement learning algorithms that were used on two different Markov Decision Process problems. The reinforcement learning algorithms were: Value Iteration, Policy Iteration and Q-Learning. The environment where the algorithms were applied were the Frozen Lake and Forest Management environments. The results were analyzed and the hyperparameter were tuned to optimize the algorithms.

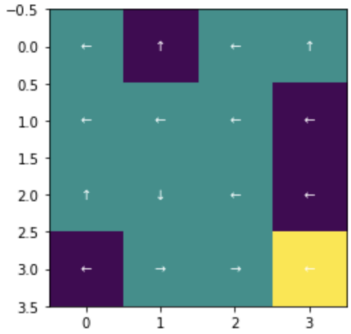

Figure 1: *Policy for Value Iteration and Policy Iteration on Frozen Lake 4x4 Environment*

II. Environments

i. The Frozen Lake Environment

The frozen lake environment is set like a grid world problem. The objective of the agent is to start at the top left block on the grid and the traverse to the bottom right of the grid to the finish line. The frozen lake environment has block where the ice is cracked and the agent can fall through and lose the game. If the agent falls through the crack, they are rewarded 0 points for the episode and if the agent is able to successfully reach the finish, the agent is rewarded 1 point.

Below is an example of the frozen lake environment with the optimal policy for a particular algorithm superimposed. In the frozen lake environment, the yellow block indicates the finish line and the purple block indicate the cracks in the ice. The green block are where the agent can traverse.

The agent has four actions which it can take: UP, DOWN, LEFT, RIGHT. This environment is stochastic, hence the agent only has a probability of 0.33 of moving in the intended direction and a probability of 0.33 of moving in the immediate clockwise and counter-clock wise directions. Two dimensions for the environment was used. A $4x4$ grid and a $16x16$ grid.

ii. The Forest Management Environment

The other environment that was used was the forest management environment. Unlike the Frozen Lake environment, the Forest Management environment is not set on a grid. The objective is to maximize the points from maintaining a forest. The ways points are rewarded are by either cutting the trees and selling the wood or by maintain the forest. Hence, there are two actions in the environment: CUT and WAIT. There are also occasional wildfires which occur with the probability of p.

The age of the tree is represented by states $0, 1...S-1$ where $S-1$ is the oldest. The WAIT action is represented by 0 and CUT action is represented by 1. After a burn or cut, the state is return to 0.

The transition and reward matrices are returned from the forest initialization function. The transition function is a three dimensional array of the shape $AxSxS$ where A are actions

and S are states. Here is how a transition matrix is defined.

```
             | p 1-p 0.......0  |
             | .  0 1-p 0....0  |
P[0,:,:] =   | .   .  0  .      |
             | .   .        .   |
             | .   .         1-p|
             | p   0  0....0 1-p|

             | 1 0..........0 |
             | . .            |
P[1,:,:] =   | . .            |
             | . .            |
             | . .            |
             | 1 0..........0 |
```

Figure 2: *Transition Matrix*

The reward function is a two dimensional array of shape SxA. Here is how a reward matrix is defined.

```
          |  0  |
          |  .  |
R[:,0] =  |  .  |
          |  .  |
          |  0  |
          |  r1 |

          |  0  |
          |  1  |
R[:,1] =  |  .  |
          |  .  |
          |  1  |
          |  r2 |
```

Figure 3: *Reward Matrix*

III. Frozen Lake 4x4 Environment

i. Value Iteration

The first algorithm that was applied on the 4x4 Frozen Lake environment was the value iteration algorithm. The discount rate and epsilon hyperparameters were tuned to optimize the algorithm. The discount rate that were used were $0.7, 0.8, 0.9, 0.95, 0.99$ and the epsilon values that were used were $1e-3, 1e-6, 1e-9, 1e-12, 1e-15$. Each of the resulting policy was tested on the environment up to 1000 episodes and the average reward was plotted.

i.1 Discount Rate

Below is the plot of the hyperparameter tuning of the discount rate. The average is shown using the blue line, the standard deviation is shown by the blue shading and the max value is shown by the yellow dashed line.

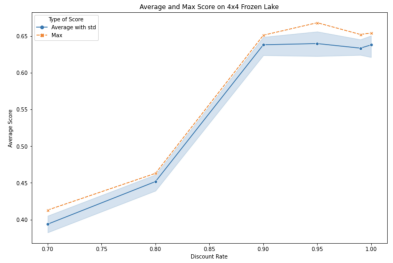

Figure 4: *Rewards for Value Iteration on Frozen Lake 4x4 Environment*

As seen from the above graph, the algorithm performs much better for discount rates above 0.9. This is due to the fact that as the states get further and further away from the terminal state, for lower values of the discount rate, it is harder to cascade the accurate values out. The policy with the optimized discount rate hyperparameter arrives at the finish 75% of the time.

The policy the this algorithm generated is showed in Figure 1. This shows that since there stochasticity and there is such a high chance of moving to the left or right block, the agent always tried to move away from the cracks in the ice.

Below are the comparisons of the time for convergence and the number of iterations taken for the different discount rates. Both graphs are similar, hence the graph for iteration is provided.

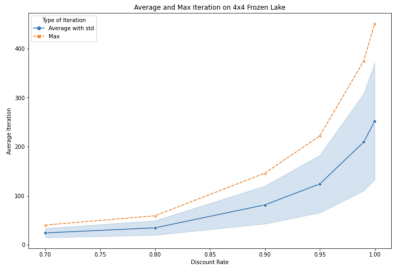

Figure 5: *Iterations per Discount Rate for Value Iteration on Frozen Lake 4x4 Environment*

As seen in the graphs, higher discount rates

yield higher number of iterations as well as higher time spent for convergence. This makes sense as there is more information cascaded out if the discount rate is higher and hence, there will be a higher difference after each episode compared to lower discount rates. However, the maximum time spent is still a quarter of a second, hence, it is worth while to use a higher discount rate.

i.2 Epsilon

Below is the plot of the hyperparameter tuning of the epsilon value. The epsilon values and graphed in a logarithmic fashion.

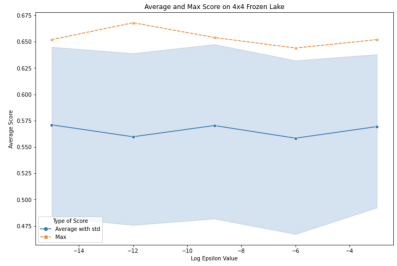

Figure 6: *Rewards for Value Iteration on Frozen Lake* 4*x*4 *Environment*

As seen from the above graph, the epsilon hyperparameter does not have much of an affect on the rewards achieved by the policy. There is a large standard deviation however which goes to show that other hyperparameters such as the discount rate does play a significant role in optimizing the policy. It seems as though the epsilon value of $1e-9$ has the highest average reward.

Below are the comparisons of the time for convergence and the number of iterations taken for the different epsilon values. Both graphs are similar, hence the graph for iteration is provided.

As seen in the graphs, higher epsilon values yield lower number of iterations as well as higher time spent for convergence. This makes sense as if the value does not have to change as much to convergence, then the policy will converge quicker. It seems as though the iterations needed for convergence increase logarithmically as the epsilon value decreases. However, the maximum time spent is still a quarter of a second, hence, it is worth while to use a higher discount rate.

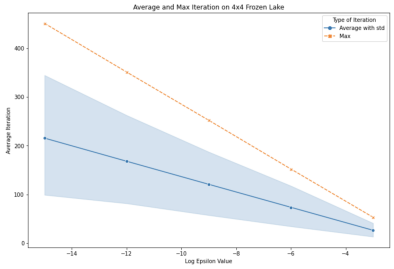

Figure 7: *Iterations per Epsilon for Value Iteration on Frozen Lake* 4*x*4 *Environment*

ii. Policy Iteration

The second algorithm that was applied on the 4*x*4 Frozen Lake environment was the value iteration algorithm. The same hyperparameters, discount rate and epsilon were tuned to optimize the algorithm. The same values for each hyperparameter was used. Each of the resulting policy was tested on the environment up to 1000 episodes and the average reward was plotted.

ii.1 Discount Rate

Below is the plot of the hyperparameter tuning of the discount rate.

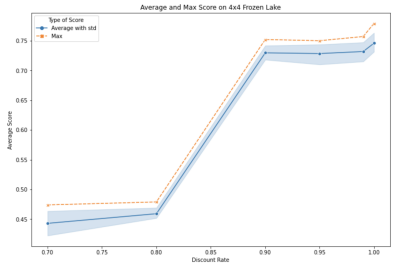

Figure 8: *Rewards for Policy Iteration on Frozen Lake* 4*x*4 *Environment*

Like for values iteration, the policy iteration generated policy performs much better for discount rates above 0.9. This is due to the fact

that for lower values of the discount rate, the values are propagated less and hence for a constant epsilon, the convergence is faster. The policy with the optimized discount rate hyperparameter arrives at the finish 75% of the time which is similar to value iteration. The policy iteration also generated the same policy as value iteration.

Below are the comparisons of the time for convergence and the number of iterations taken for the different discount rates.

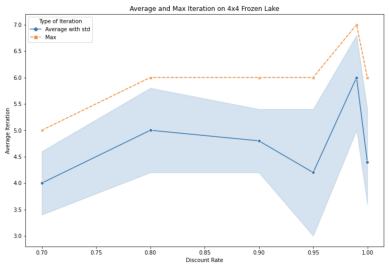

Figure 9: *Iterations per Discount Rate for Policy Iteration on Frozen Lake* 4*x*4 *Environment*

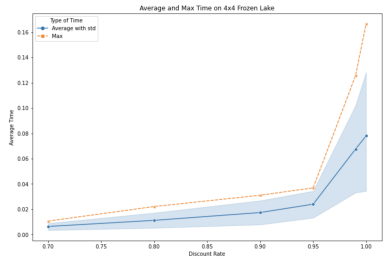

Figure 10: *Time per Discount Rate for Policy Iteration on Frozen Lake* 4*x*4 *Environment*

Unlike for value iteration, the number of iteration or episodes needed for policy iteration to converge is between 3 and 6 on average. This is compared to value iteration taking up to 1200 iterations. But, similar to value iteration, higher discount rates yield higher number of time spent for convergence. However, it does seem that policy iteration takes less time than value iteration. The maximum time taken by any policy is about 0.22 for policy iteration compared to 0.25 for values iteration and the average time for the highest discount rate for policy iteration is 0.12 and for value iteration is 0.14. Although the difference is slight for a small grid, it would be noticeable for a larger grid.

ii.2 Epsilon

Below is the plot of the hyperparameter tuning of the epsilon value. The epsilon values and graphed in a logarithmic fashion.

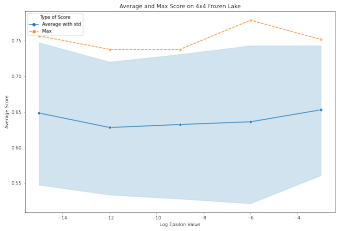

Figure 11: *Rewards for Policy Iteration on Frozen Lake* 4*x*4 *Environment*

Similar to value iteration, the epsilon hyperparameter does not have much of an affect on the rewards achieved by the policy. There is a large standard deviation however which goes to show that other hyperparameters such as the discount rate does play a significant role in optimizing the policy. It seems as though the epsilon value of $1e-12$ has the highest average reward.

Below are the comparisons of the time for convergence and the number of iterations taken for the different epsilon values.

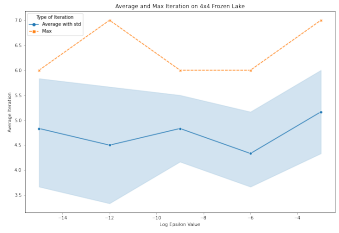

Figure 12: *Iterations per Epsilon for Policy Iteration on Frozen Lake* 4*x*4 *Environment*

Similar to the graph for discount rate, the number of iteration on average taken for the

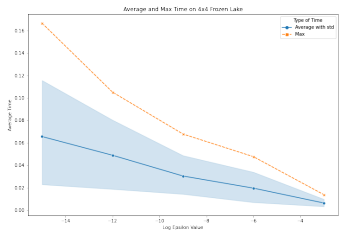

Figure 13: *Time per Epsilon for Policy Iteration on Frozen Lake* 4*x*4 *Environment*

different epsilons are between 3 and 6. Hence it seems that that metric is random.

iii. Q-Learning

There were three hyperparameters that were tested and analyzed for the Q-Learning algorithm. The three hyperparameters were: number of episodes, epsilon decay rate and the learning rate.

The first hyperparameter that was tested was the learning rate. The two learning rates that were tested were 0.1 and 0.01. The epsilon, epsilon decay and discount rates were held constant. The graphs for the rolling average of the rewards for each of the experiments are shown below. The x-axis is log normal distribution.

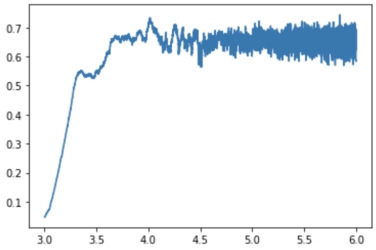

Figure 14: *Rolling Average of Rewards for Learning Rate* 0.1 *for Q-Learning on Frozen Lake* 4*x*4 *Environment*

It can be seen from the above graphs that the a larger learning rate leads to the faster convergence. However, it can be seen that for a larger learning rate, there is more oscillation towards the end of the learning process. This is due to the fact that the reward for each episode

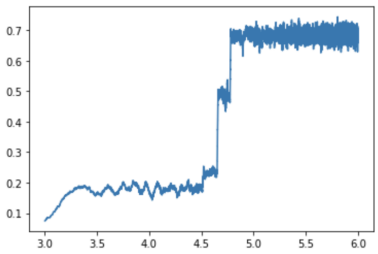

Figure 15: *Rolling Average of Rewards for Learning Rate* 0.01 *and Epsilon Decay* $1e-3$ *for Q-Learning on Frozen Lake* 4*x*4 *Environment*

affects the Q-table more significantly. Hence, it is appropriate to have an decay on the learning rate as well. If a constant learning rate is used, one should be chosen that balances the speed in convergence and minimizes oscillation.

The second hyperparameter that was tuned was epsilon decay. Since it was found that the learning rate of 0.01 had less oscillation, that was chosen as the constant. The two epsilon decay rates were $1e-3$ and $1e-5$. The graph for the rolling average of the rewards for the former graph is above and the later is below:

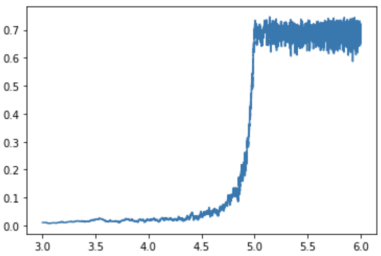

Figure 16: *Rolling Average of Rewards for Epsilon Decay* $1e-5$ *for Q-Learning on Frozen Lake* 4*x*4 *Environment*

The graphs of the epsilon decay seems pretty similar since the x-axis is log normal distributed, however, the experiment with a higher decay converges faster than the one with lower decay. The epsilon decay of $1e-3$ reached convergence at 50000 iteration whereas the epsilon decay of $1e-5$ reaches convergence at 100000 iterations. The reason the the decay rate did not have a large affect on the convergence is because of the stochasticity. If the

stochasticity was lower, there would a larger difference in the convergence of the different decay rates.

The last hyperparameter that was tested was the number of training episodes. Below is the chart of the performance for Q-Learning using the different number of episodes, epsilon decay rate and the learning rate.

	Discount Rate	Training Episodes	Learning Rate	Decay Rate	Reward	Time Spent
0	0.9999	10000.0	0.10	0.00100	0.717	14.662740
1	0.9999	10000.0	0.10	0.00001	0.748	1.995275
2	0.9999	10000.0	0.01	0.00100	0.108	2.631589
3	0.9999	10000.0	0.01	0.00001	0.317	2.001463
4	0.9999	100000.0	0.10	0.00100	0.710	156.078970
5	0.9999	100000.0	0.10	0.00001	0.742	61.301273
6	0.9999	100000.0	0.01	0.00100	0.129	98.832058
7	0.9999	100000.0	0.01	0.00001	0.736	59.623418
8	0.9999	1000000.0	0.10	0.00100	0.757	1949.194673
9	0.9999	1000000.0	0.10	0.00001	0.732	1721.441356
10	0.9999	1000000.0	0.01	0.00100	0.704	1491.809788
11	0.9999	1000000.0	0.01	0.00001	0.712	1463.943181

Figure 17: *Chart for Q-Learning on Frozen Lake* 4*x*4 *Environment*

It can be seen that the number of training episodes had a large affect on the time spent on learning. Q-Learning takes significantly longer to learn than value or policy iteration. This is the case because Q-Learning is a model free learning algorithm and has to explore the environment to generate the model. It seems as though the highest possible reward is 0.747 which was able to be achieved by several of the policies. It seems as though having higher learning rate and epsilon decay rates help with achieving higher rewards. It can also be seen that with a larger number of iteration, the average reward regardless of learning rate or epsilon decay rate increases.

Below is the policy that was determined by the Q-Learning algorithm. This is slightly different that than found by value iteration and policy iteration.

IV. Frozen Lake 16*x*16 Environment

The three learning algorithms were then tested on a larger 16*x*16 grid frozen lake environment. Since the rewards for falling in a crack and finishing are the same and the grid is larger, the average reward will be lower.

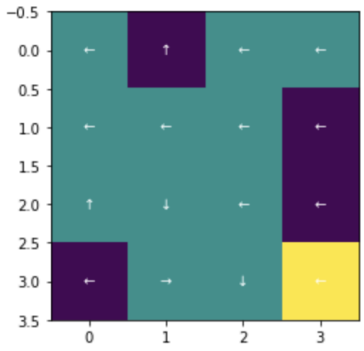

Figure 18: *Policy for Q-Learning on Frozen Lake* 4*x*4 *Environment*

i. Value Iteration

The first algorithm which was tested was value iteration. The discount rate hyperparameter was tuned to see if there was a difference for a larger grid. Indeed, from the chart below, it can be seen that the even for large discount rates above 0.9, the average reward is quite low and there is exponential growth in the average reward as the discount rate approaches 1. This shows that for larger grids, it is better to use larger discount rates.

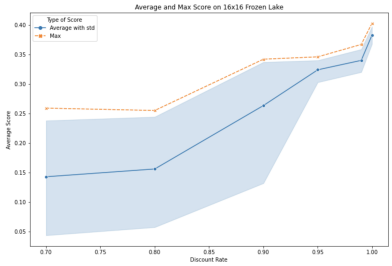

Figure 19: *Rolling Average of Rewards for Discount Rate for Value Iteration on Frozen Lake* 16*x*16 *Environment*

Below are the graphs for the number of iteration and length of time taken for learning each policy at different discount rates. Both graphs are similar, hence the graph for iteration is provided.

Though the shapes of the graphs are the same as for a smaller grid, it can be seen that the number of iteration and the length of time

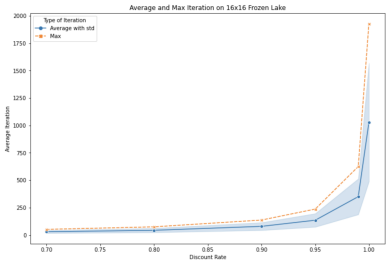

Figure 20: *Iterations per Discount Rate for Value Iteration on Frozen Lake* 4*x*4 *Environment*

for convergence have grow exponentially. The max number of iterations to learn a policy was 5000 compared to 1200 and the time spent was 14 seconds compared to 0.25 seconds.

ii. Policy Iteration

The second algorithm that was tested was policy iteration. Like for value iteration, the graph of the discount rate hyperparameter tuning has an exponential growth in the average rewards as the discount rate approaches 1. However, for policy iteration, using lower discount rates yield an higher average reward, but the maximum average reward is similar to that of value iteration.

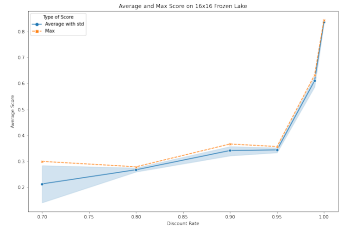

Figure 21: *Rolling Average of Rewards for Discount Rate for Policy Iteration on Frozen Lake* 16*x*16 *Environment*

Below are the graphs for the number of iteration and length of time taken for learning each policy at different discount rates.

The time taken for convergence for policy iteration is significantly longer than that for value iteration even though policy iteration has far fewer iterations. That is because for policy

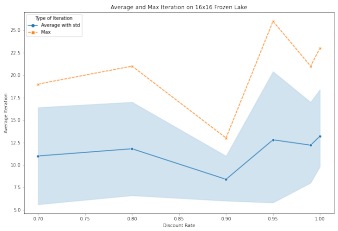

Figure 22: *Iterations per Discount Rate for Policy Iteration on Frozen Lake* 4*x*4 *Environment*

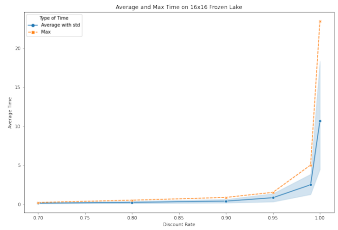

Figure 23: *Time per Discount Rate for Policy Iteration on Frozen Lake* 4*x*4 *Environment*

iteration, a large number of states need to be evaluated for each iteration. Hence, since value iteration and policy iteration have similar score for rewards and the fact that policy iteration takes longer, it is seen that value iteration is a better algorithm.

iii. Q-Learning

The third learning algorithm that was tested was Q-learning. However, even with training episodes as high as a million, none of the agents were able to successfully traverse to the goal due to exploration/exploitation. This shows that Q-Learning is not an optimal policy for the way this environment is set up. Since the environment has a model, it is better to use a model-based learning algorithm like value iteration or policy iteration.

There are several ways to tweak the environment for the agents to learn using Q-Learning. One way is to provide a different set of rewards. For example, providing a very small reward for staying alive or having a negative reward for

falling into the crack. Another way is to start the agent at random states. Then the agent would have an easier time finding the goal and populating the Q table.

V. Forest Management 20 States

For the forest management environment, the number of states will be 20 and the discount rate will be held constant at 0.9. This is because there are multiple sources of rewards. By changing the discount rate, the agent will change its behavior on when to cut the trees and hence produce different rewards. In addition the starting state during the learning process was random since if the starting state was always 0, then the policy would always be to cut. Hence the agent was started at each state 1000 time and the average reward was taken. In addition, the reward for keeping a tree at the final state was set to 6 and the reward for cutting a tree at the final state was 10.

i. Value Iteration

The epsilon hyperparameter was tuned on the value iteration learning algorithm. The epsilon values that were used were $0.1, 0.01, 1e-6, 1e-9, 1e-12, 1e-15$. Since there are rewards for multiple state, action pairs, it was easy for the value iteration algorithm to fill out the value table and come to an optimal policy for each epsilon value.

	Epsilon	Policy	Iteration	Time	Reward	Value Function
0	1.000000e-01	(0, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...	33	0.009009	2.843259	(4.328504893081768, 4.881518944971712, 4.88151...
1	1.000000e-03	(0, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...	55	0.002171	2.842526	(4.460720290173723, 5.013211594807497, 5.01321...
2	1.000000e-06	(0, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...	87	0.003533	2.920512	(4.474643139166661, 5.027129333047953, 5.02712...
3	1.000000e-09	(0, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...	120	0.004727	2.884589	(4.475122925121185, 5.027609012960728, 5.02760...
4	1.000000e-12	(0, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...	153	0.006271	2.885044	(4.475137648838066, 5.027623839684378, 5.02762...
5	1.000000e-15	(0, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...	186	0.009037	2.913634	(4.475138109938985, 5.027624294784101, 5.0276...

Figure 24: *Average of Rewards for Epsilon for Value Iteration on Forest Management* 20 *State Environment*

The policies returned by by each of the algorithms are the same and the rewards are the similar given the variance in the start state. It can be seen that larger epsilon value algorithms take less iterations and time, hence they are more optimal.

ii. Policy Iteration

Since the mdptoolbox implementation of policy iteration was used, the epsilon hyperparameter was not able to be tuned. However the policy return by the algorithm was the same as it was for value iteration, hence the reward was statistically similar. However, the policy iteration algorithm took 12 iterations and 0.003 second to run to completion which is longer than the time taken for value iteration. This is different than in the case for the $4x4$ Frozen Lake environment because policy iteration changes a smaller part of the policy due to the chained state set-up environment unlike for the frozen lake environment where policy iteration is able to change the policy on a large part of the grid.

iii. Q-Learning

Since in Q-Learning the agent has to balance exploration and exploitation in an environment where there are lots of rewards possible, it seems hard for the agent to learn the optimized policy. There were 5 hyperparameter that were tested: number of iterations, epsilon, epsilon decay, learning rate and learning rate decay.

Like for value iteration and policy iteration, the rewards using Q-Learning were similar. However, the policies generated by the algorithms were different. In fact, none of the policies generated by Q-Learning matched that generated by value iteration and policy iteration. In addition, the time spent learning was significantly longer for Q-Learning. It took up to 10 minutes whereas the longest it took for previous algorithms was 0.03 second.

VI. Forest Management 500 States

In order to compare and contrast a small forest management environment versus a large environment, an environment with 500 states was tested. The discount rate was kept constant at 0.99 and the reward for cutting a tree at the final state was set to 100 and the the reward for maintaining a tree at the final state was set to 15.

	Iterations	Alpha Decay	Alpha Min	Epsilon	Epsilon Decay	Reward	Time
0	1000000	0.990	0.0010	10.0	0.990	3.241963	31.702035
1	1000000	0.990	0.0001	10.0	0.990	3.454687	31.296608
2	1000000	0.999	0.0010	10.0	0.990	2.972591	32.328044
3	1000000	0.999	0.0001	10.0	0.990	3.119909	31.661542
4	1000000	0.990	0.0010	10.0	0.999	1.100000	31.660677
5	1000000	0.990	0.0001	10.0	0.999	3.460559	31.859090
6	1000000	0.999	0.0010	10.0	0.999	3.195792	31.749441
7	1000000	0.999	0.0001	10.0	0.999	0.800000	31.852308
8	1000000	0.990	0.0010	1.0	0.990	3.319346	31.965991
9	1000000	0.990	0.0001	1.0	0.990	3.358040	31.957316
10	1000000	0.999	0.0010	1.0	0.990	3.052964	31.986225
11	1000000	0.999	0.0001	1.0	0.990	3.165224	31.556049
12	1000000	0.990	0.0010	1.0	0.999	3.405795	31.564664
13	1000000	0.990	0.0001	1.0	0.999	3.491387	31.755743
14	1000000	0.999	0.0010	1.0	0.999	3.200832	31.798701
15	1000000	0.999	0.0001	1.0	0.999	3.140348	31.862554
16	10000000	0.990	0.0010	10.0	0.990	3.369754	319.295502
17	10000000	0.990	0.0001	10.0	0.990	3.334106	321.230147
18	10000000	0.999	0.0010	10.0	0.990	2.820745	327.029955
19	10000000	0.999	0.0001	10.0	0.990	0.850000	330.888483
20	10000000	0.990	0.0010	10.0	0.999	0.950000	323.813685
21	10000000	0.990	0.0001	10.0	0.999	3.486292	322.142016
22	10000000	0.999	0.0010	10.0	0.999	3.271430	319.755924
23	10000000	0.999	0.0001	10.0	0.999	0.950000	320.945358
24	10000000	0.990	0.0010	1.0	0.990	3.139218	322.003368
25	10000000	0.990	0.0001	1.0	0.990	3.404987	322.042986
26	10000000	0.999	0.0010	1.0	0.990	3.114508	322.882178
27	10000000	0.999	0.0001	1.0	0.990	2.935014	322.681150
28	10000000	0.990	0.0010	1.0	0.999	3.455509	322.382801
29	10000000	0.990	0.0001	1.0	0.999	3.416639	321.686291
30	10000000	0.999	0.0010	1.0	0.999	3.322320	320.345625
31	10000000	0.999	0.0001	1.0	0.999	1.000000	317.431135

Figure 25: *Rewards for Hyperparameter Tuning for Q-Learning on Forest Management* 20 *State Environment*

i. Value Iteration

The epsilon hyperparameter was tuned for the large forest management environment as well. The same values for epsilon were used: $0.1, 0.01, 1e-6, 1e-9, 1e-12, 1e-15$.

As for the smaller environment, the average reward for all the policies generated with different epsilon values were extremely close. This means that even a large epsilon value such as 0.1 is small enough to converge to the optimal policy. The number of iteration taken for convergence was about double that for the 20 state

	Epsilon	Policy	Iteration	Time	Reward
0	1.000000e-01	(0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, ...	79	0.008641	2.725382
1	1.000000e-03	(0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, ...	119	0.010274	2.726020
2	1.000000e-06	(0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, ...	179	0.015920	2.750285
3	1.000000e-09	(0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, ...	239	0.023410	2.755154
4	1.000000e-12	(0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, ...	299	0.026353	2.735265
5	1.000000e-15	(0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, ...	349	0.031340	2.743204

Figure 26: *Average of Rewards for Epsilon for Value Iteration on Forest Management* 500 *State Environment*

environment and the time taken grew linearly which shows that value iteration is good learning algorithm for larger environments. This pattern was also seen in the frozen lake environments.

ii. Policy Iteration

The policy iteration algorithm was able to get an average reward of 2.72 which is similar to that of value iteration. However, the policy iteration algorithm took 3.1 second to train which is about 8 times longer than that for value iteration. This is due to the fact that the policies are dependent on each other in the policy iteration algorithm. When the policy of the next state changes, that affects the policy of the current state and hence more iteration and time is needed. This is similar to that seen in the frozen lake environment where for large environments, policy iteration take much longer than value iteration.

iii. Q-Learning

Like for the smaller environment, there were five hyperparameter tuned: number of iterations, epsilon, epsilon decay, learning rate and learning rate decay.

Unlike the previous test, the combination of different values for hyperparameter yielded by very different average rewards. Some algorithms got an average reward of below 1 while other were able to get average rewards higher than 2.8. One hyperparameter which had a large effect on the average reward was the number of iterations. This is obvious as a larger number of iterations means more training and it makes sense that a larger environ-

	Iterations	Alpha Decay	Alpha Min	Epsilon	Epsilon Decay	Reward	Time
0	1000000	0.990	0.0010	10.0	0.990	2.645720	48.420391
1	1000000	0.990	0.0001	10.0	0.990	2.649980	48.929765
2	1000000	0.999	0.0010	10.0	0.990	2.639288	48.745058
3	1000000	0.999	0.0001	10.0	0.990	2.631506	48.816628
4	1000000	0.990	0.0010	10.0	0.999	2.633484	48.870897
5	1000000	0.990	0.0001	10.0	0.999	2.662812	50.872312
6	1000000	0.999	0.0010	10.0	0.999	2.579536	49.520571
7	1000000	0.999	0.0001	10.0	0.999	2.640828	48.303403
8	1000000	0.990	0.0010	1.0	0.990	2.615334	48.022415
9	1000000	0.990	0.0001	1.0	0.990	0.854000	47.887792
10	1000000	0.999	0.0010	1.0	0.990	2.625327	48.091998
11	1000000	0.999	0.0001	1.0	0.990	2.622870	47.902281
12	1000000	0.990	0.0010	1.0	0.999	2.641285	48.488182
13	1000000	0.990	0.0001	1.0	0.999	2.638887	48.764838
14	1000000	0.999	0.0010	1.0	0.999	2.627288	48.332479
15	1000000	0.999	0.0001	1.0	0.999	2.621154	49.929556
16	10000000	0.990	0.0010	10.0	0.990	2.753958	483.689740
17	10000000	0.990	0.0001	10.0	0.990	2.830737	473.040089
18	10000000	0.999	0.0010	10.0	0.990	2.770383	479.502859
19	10000000	0.999	0.0001	10.0	0.990	2.800512	478.104773
20	10000000	0.990	0.0010	10.0	0.999	2.759782	475.597373
21	10000000	0.990	0.0001	10.0	0.999	2.838975	468.015625
22	10000000	0.999	0.0010	10.0	0.999	2.738463	471.800302
23	10000000	0.999	0.0001	10.0	0.999	2.805521	463.277349
24	10000000	0.990	0.0010	1.0	0.990	2.750630	465.241464
25	10000000	0.990	0.0001	1.0	0.990	2.821698	460.766587
26	10000000	0.999	0.0010	1.0	0.990	2.758731	466.607251
27	10000000	0.999	0.0001	1.0	0.990	2.789471	467.453709
28	10000000	0.990	0.0010	1.0	0.999	2.762796	469.504692
29	10000000	0.990	0.0001	1.0	0.999	2.831903	463.713984
30	10000000	0.999	0.0010	1.0	0.999	2.766255	519.160908
31	10000000	0.999	0.0001	1.0	0.999	2.781121	628.062878

Figure 27: *Rewards for Hyperparameter Tuning for Q-Learning on Forest Management* 500 *State Environment*

ment would need more training. 10 million iterations seems sufficient for convergence.

Iterations	Alpha Decay	Alpha Min	Epsilon	Epsilon Decay	Reward	Time
1000000	0.9945	0.00055	5.5	0.9945	2.520581	48.743660
10000000	0.9945	0.00055	5.5	0.9945	2.785058	483.346224

Figure 28: *Rewards for Iterations for Q-Learning on Forest Management* 500 *State Environment*

However, the biggest indicator of performance was the learning rate decay. It can be seen from the below graph that ta higher learning rate decay leads to a slower decrease in the learning and hence a better performance. This is due to the fact that is the learning rate decreases too quickly, the agent will not be able to explore the state space sufficiently and the q table will not be updated significantly is later iterations.

Epsilon Decay	Alpha Decay	Alpha Min	Epsilon	Reward	Time
0.990	0.9945	0.00055	5.5	2.597509	260.076425
0.999	0.9945	0.00055	5.5	2.708131	272.013459

Figure 29: *Rewards for Learning Rate Decay for Q-Learning on Forest Management* 500 *State Environment*

The Q-Learning algorithm does not generate the same policy as value iteration or policy iteration, however, given the optimal hyperparameters, Q-Learning perform significantly better at about an average reward of 2.86.

VII. Conclusion

After testing the value iteration, policy iteration and Q-Learning algorithms on the forest management and frozen lake environments, it can be seen the the two model-based algorithms performed better. This is due to the fact that they were able to interact with the model directly. In fact the value iteration and policy iteration algorithms did better on both the reward and time metrics. The Q-Learning algorithm which had to interact with the environment to learn performed significantly worse and was not able to learn anything in large environments. This can be changed by hyperparameter tuning and altering the environment slightly, but it always takes significantly longer.

References

[1] Richard S. Sutton *Reinforcement Learning* [*PDF*]. Kluwer Academic Publishers, Boston, 1992.