Assignment 4 (Markov Decision Processes)

Cindy Nyoumsi
snyoumsi@gatech.edu

***Abstract*—** In this report, I will be implementing two Markov Decision Process Problems, one of them being a Gridworld problem (The Frozen lake problem) and the other a non-Gridword problem (The Forest Management problem). I will then try to solve these problems using both value and policy iteration one at the time and comparing the results of my experiments. I will then try to resolve these two problems using a reinforcement learning algorithm and comparing the results once more to the other solutions using MDPs.

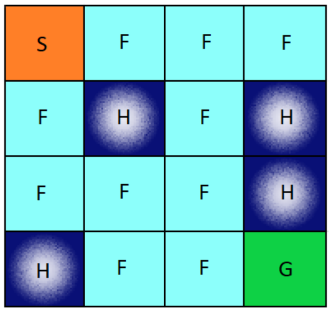
1 MDP PROBLEMS DESCRIPTION & WHY THEY ARE INTERESTING

I will be describing below the MDP problems I will be solving using both MDP approaches and reinforcement learning ;

1.1 Frozen Lake

The first MDP problem I will be solving is the Frozen Lake gridworld problem. In this MDP problem, the environment consists of a grid made up of four types of tiles, which are a starting tile shown as 'S' in the image below, frozen tiles shown as 'F' below, hole tiles shown as 'H' and the goal tile shown as 'G', with an agent that is trying to obtain a reward by reaching the goal tile by following an optimal path that avoids the hole tiles. The movements the agent is allowed are Up, Left, Right, Down.

I found this problem interesting because it is a great example of a markov decision process given that it meets all the requirements of what makes an MDP from having states as represented by where on the grid it will be, and the ability to accomplish goals the reaching the goal tile with rewards and following a specific policy and can be made stochastic or non-stochastic.



1.2 Forest Management

The second MDP problem I will be solving is the Forest Management problem. In this MDP problem, the environment consists of a forest that contains wildlife and the agent's role in this forest is to maintain wildlife while making money from cutting trees. The states in this problem correspond to the different age groups the trees have been clustered into from ages 0 - 20, 21 - 40, and 41+ in that order for each state, with rewards of 0 for cutting at stage 1, reward 1 for cutting at stage 2 and reward of 2 for cutting at stage 3. The goal being here to maximize the rewards while not waiting too long and experiencing a possible fire.

I found this problem to be even more interesting than the frozen lake problem since this is a real world planning problem and is more complex and also is naturally stochastic since we can't control the occurrence of a wildfire which will bring us back to our initial state.

2 FROZEN LAKE PROBLEM

I resolved the Frozen lake problem in two different states. The first one was a 4x4 grid and the second was a 16x16 grid.

2.1 Frozen Lake Value Iteration vs Policy Iteration

2.1.1 Frozen Lake 4X4 Grid (16 States)

In this section I will be discussing how many iterations it takes for each policy to converge as well as which policies converge faster and to which scores and finally also looking at how the number of states affects convergence and performance.

Before going further, when discussing the concept of convergence for the algorithms below, convergence in these cases is defined as the value after which further iterations return the same value with only a very small delta corresponding to a difference in decimal places such that if rounded, the value for those iterations rounds to the same number.

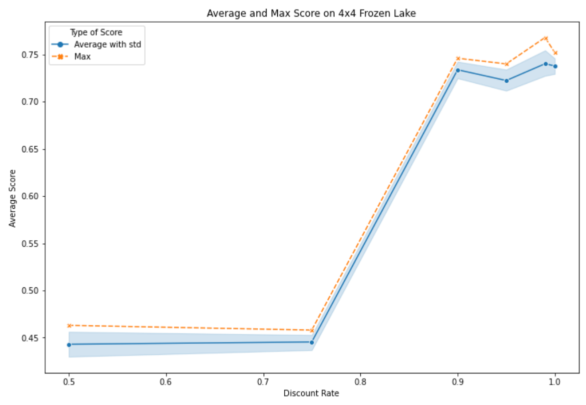
Starting with the 16 state frozen lake problem, it took about ***700 iterations to converge 74% percent of the time using value iteration,*** while it took only ***7 iterations to converge 76% of the time with policy iteration*** for the same states, which makes sense since policy iteration technically has less outer loops in its algorithm than Value iteration does.

It took ***0.05 seconds*** to converge with value iteration while it took only ***0.02 seconds*** to converge for the policy iteration. This again makes sense given that it also took a lot more iterations and that the value iteration algorithm has one extra layer than the policy iteration algorithm.

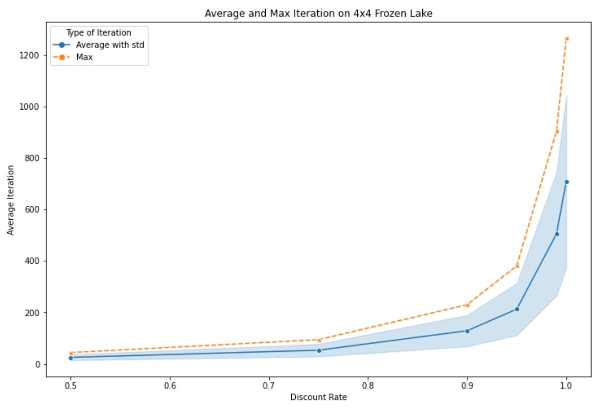
It was also interesting to see that as the discount rate increased, using value iteration, the number of iterations required to converge also increased exponentially, which also overall increased the probability of reaching the goal at a higher frequency while for policy iteration because of the more stochastic nature of the policy algorithm, we see that it unexpectedly rises linearly before dropping again as the discount score increases, although overall in both cases, a higher discount rate results in a higher reward.

See charts of average number of iterations against discount value and average score against discount value below.

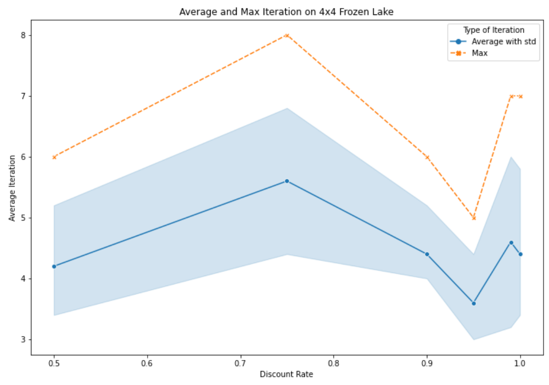
Average # of Value Iterations vs Discount Rate



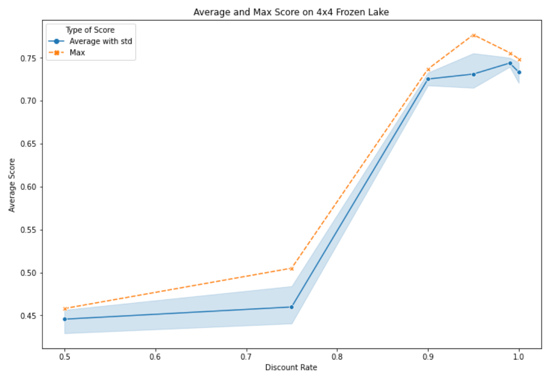
Value Iteration Average score vs Discount Rate



Average # of Policy Iterations vs Discount Rate



Policy Iteration Average score vs Discount Rate



2.1.2 Frozen Lake 16X16 Grid (256 States)

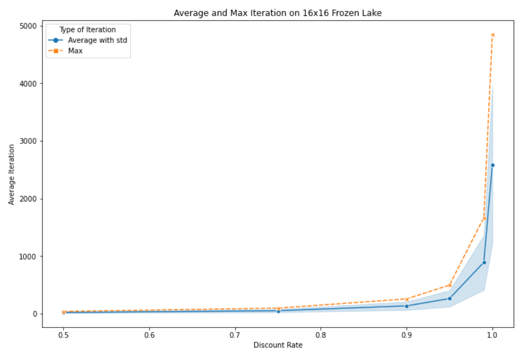
For the 256 state frozen lake problem, it took about ***1600 iterations to converge ~47% percent of the time using value iteration,*** while it took only ***20 iterations to converge ~51% of the time with policy iteration*** for the same states. The gap between value and policy iteration has widened even more as we increase the number of states. Compared to the 16 state frozen lake problem, we see that the number of iterations only doubled despite the number of states quadroubling, while the reward decreased given that the path to the goal tile was longer therefore increasing the probability of going into a hole tile and decreasing expected return overall.

A very interesting observation here is that It merely took ***3 seconds*** to converge with value iteration while it took ~***37 seconds*** to converge for the policy iteration. Which is complete opposite of expected performance given there is a much higher number of iterations for value iteration, but upon looking at each algorithm closer, we can see that policy iteration has one more step in the loop and thus for larger number of states the calculation is more expensive as we have to evaluate both the policy and value for each additional iteration thus increasing the runtime.

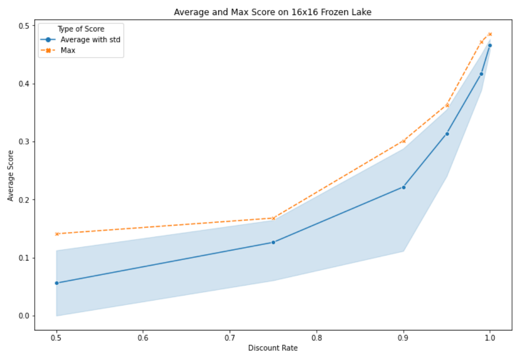
We can also notice that it took about double the number of iterations in both cases even though the number of states got squared. This means that for how much the states increased the number of iterations did not increase as much which is a good thing when thinking about using these algorithms for problems with very large states.

See charts of average number of iterations against discount value and average score against discount value below.

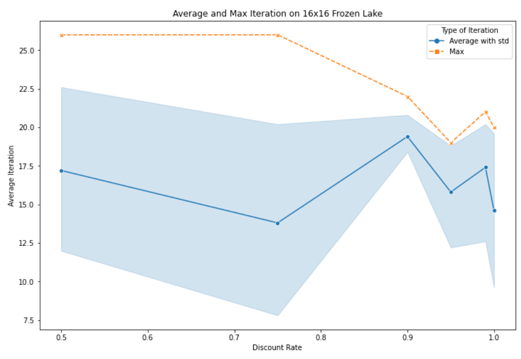
Average # of Value Iterations vs Discount Rate



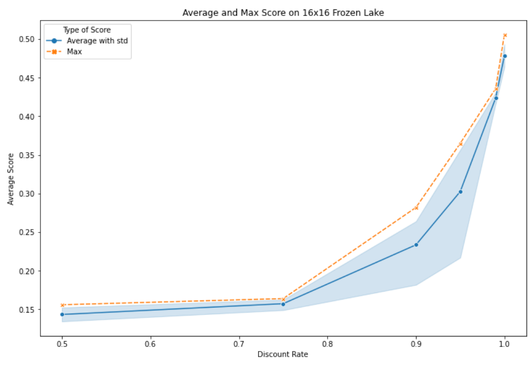
Value Iteration Average score vs Discount Rate



Average # of Policy Iterations vs Discount Rate



Policy Iteration Average score vs Discount Rate



2.2 Frozen Lake Reinforcement Learning with Q-Learning

I decided to use Q-Learning as my learning choice for solving the Frozen Lake and Forest Management problems with Reinforcement Learning.

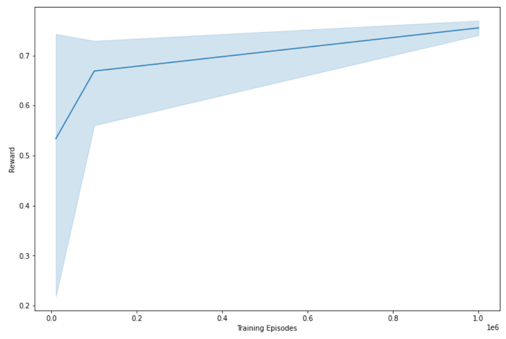
2.2.1 Frozen Lake 4X4 Grid (16 States)

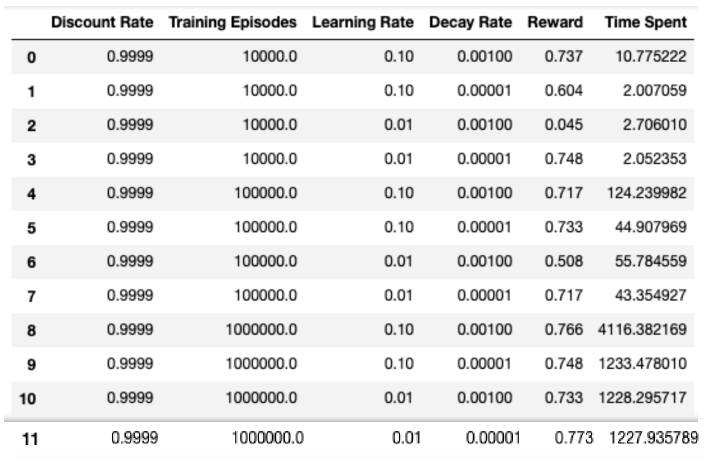
For the 16 state grid Frozen Lake problem, Q-Learning performed **about the same** in terms of average reward achieved as we increased the number of training episodes it learned from, at a 0.01 learning rate getting as high as **0.773 reward** which was higher than the convergence reward achieve with both value and policy iteration. This outcome is understandable when we think about the fact that the agent in Q-learning is figuring out the policy and value function by experience and does not know what the rewards are for each move in advance, but it gets better as it gains more experience through training episodes, thus improving its reward, but at the expense of a higher run time and iterations. We can see in the chart below that as the number of training episodes increased the agent's reward increased as it learned more.

Timewise it took on average a lot longer to learn compared to the microseconds we saw with policy and value iteration, it took 1227 seconds for its best reward on 1,000,000 training episodes.

In terms of exploration strategy I adjusted the learning rate, number of training episodes and decay rate. As shown in the table below, we can see that the agent achieves the highest reward with one million training episodes, and at a learning rate of 0.01 and for a decay rate of 0.0001. I think this combination worked best because the high decay rate and low learning rate balances each other out and the number of training examples is large enough for the agent to explore as much of the environment as possible.

Reward vs Training Episodes





	Discount Rate	Training Episodes	Learning Rate	Decay Rate	Reward	Time Spent
0	0.9999	10000.0	0.10	0.00100	0.737	10.775222
1	0.9999	10000.0	0.10	0.00001	0.604	2.007059
2	0.9999	10000.0	0.01	0.00100	0.045	2.706010
3	0.9999	10000.0	0.01	0.00001	0.748	2.052353
4	0.9999	100000.0	0.10	0.00100	0.717	124.239982
5	0.9999	100000.0	0.10	0.00001	0.733	44.907969
6	0.9999	100000.0	0.01	0.00100	0.508	55.784559
7	0.9999	100000.0	0.01	0.00001	0.717	43.354927
8	0.9999	1000000.0	0.10	0.00100	0.766	4116.382169
9	0.9999	1000000.0	0.10	0.00001	0.748	1233.478010
10	0.9999	1000000.0	0.01	0.00100	0.733	1228.295717
11	0.9999	1000000.0	0.01	0.00001	0.773	1227.935789

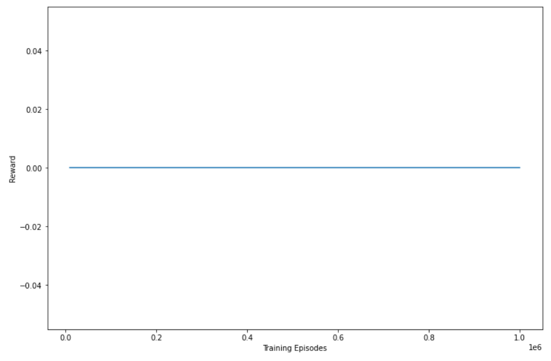
2.2.2 Frozen Lake 16X16 Grid (256 States)

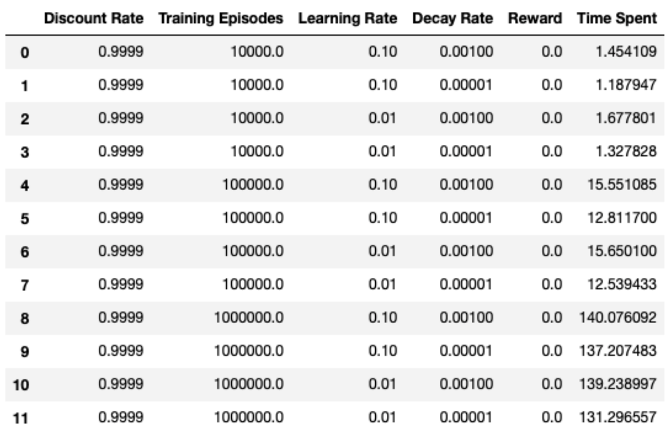
Extremely interesting observation for the 256 state grid Frozen Lake problem, Q-Learning performed quite terribly with the agent never reaching the goal and thus constantly achieving a reward of 0. This makes sense given that the expected reward decreases as we increase the number of states as we have seen earlier with the VI and PI performing worst on larger states, but in this case, because our agent is exploring the environment randomly it increases the probability of ending up in a failure and never reaching the goal since there is only one goal.

It also took a lot of time on average to explore the larger environment with the maximum runtime at 131 seconds which is still quite a significantly smaller amount of time than it took to reach the highest reward in the 16 state frozen lake problem.

I used the same exploration strategy as with the 16 state problem and basically adjusted the learning rate, number of training episodes and decay rate. As shown in the table below, we can see that other than increasing the training time these parameters don't change the fact that the agent is unable to reach the goal. This means that this type of problem is not well suited for reinforcement learning and PI/VI are better algorithms for it.

Reward vs Training Episodes





	Discount Rate	Training Episodes	Learning Rate	Decay Rate	Reward	Time Spent
0	0.9999	10000.0	0.10	0.00100	0.0	1.454109
1	0.9999	10000.0	0.10	0.00001	0.0	1.187947
2	0.9999	10000.0	0.01	0.00100	0.0	1.677801
3	0.9999	10000.0	0.01	0.00001	0.0	1.327828
4	0.9999	100000.0	0.10	0.00100	0.0	15.551085
5	0.9999	100000.0	0.10	0.00001	0.0	12.811700
6	0.9999	100000.0	0.01	0.00100	0.0	15.650100
7	0.9999	100000.0	0.01	0.00001	0.0	12.539433
8	0.9999	1000000.0	0.10	0.00100	0.0	140.076092
9	0.9999	1000000.0	0.10	0.00001	0.0	137.207483
10	0.9999	1000000.0	0.01	0.00100	0.0	139.238997
11	0.9999	1000000.0	0.01	0.00001	0.0	131.296557

3 FOREST MANAGEMENT PROBLEM

I resolved the Forest Management problem in two different states. The first one was with 20 states and the second was with 500 states.

Like I mentioned earlier, this MDP problem is a lot more interesting than the Frozen Lake problem because there is more than just one goal that is being rewarded. This also makes this problem more complex.

3.1 Forest Management Value Iteration vs Policy Iteration

3.1.1 Forest Management 20 States

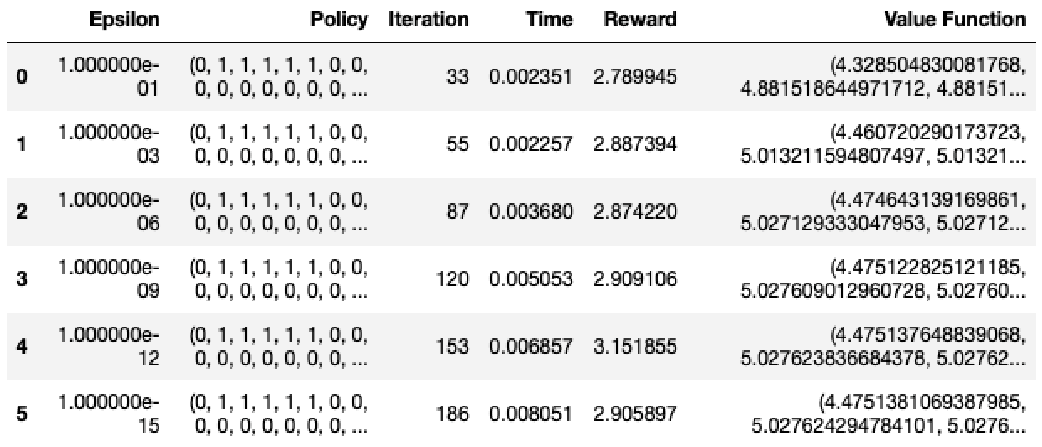
Starting with the 20 state Forest Management problem, it was a little more complex to set up unlike the FrozenLake problem.

For the value iteration, instead of having multiple iterations with the same start and watching for convergence, I instead ran the model for a fixed number of times for each parameter combination I was using. In this case I was running value iteration over six different epsilon values and evaluating the expected reward for 1,000,000 value iterations for each of the set ups. Focusing on the best results, the convergence happening at the best ***one millionth value iteration*** had an expected convergence **value of 3.15**.

For the policy iteration, given that I could use the mdp built in module's functions, it was easier to evaluate the convergence for the policy iteration which was ***2.72*** after ***14 iterations only***, taking ***0.0071 seconds*** only. The convergence value with policy iteration was quite lower than with policy iteration but still within the ballpark and at higher efficiency. The reason why policy iteration is so much more efficient for this non-grid world problem is because there is an order dependency in the policy so instead of evaluating the value for multiple states, we are focusing on one state at a time. The same reasoning goes for why value iteration is much more efficient in this problem as well despite the higher complexity of the problem.

It merely took ***0.0006 seconds*** to converge with value iteration while it took ***0.007 seconds*** to converge for the policy iteration. This was a little surprising to me given that policy iteration was so much faster when we were working with Grid Problems, yet now we have more complex MDP problems, it takes more time to iterate over the entire policy which makes sense. See table below for 6 epsilon parameter models for value iteration implementations.

Value Iteration Results Table



	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	0.002351	2.789945	(4.328504830081768, 4.881518644971712, 4.88151...
1	1.000000e-03	(0, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...	55	0.002257	2.887394	(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	0.003680	2.874220	(4.474643139169861, 5.027129333047953, 5.02712...
3	1.000000e-09	(0, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...	120	0.005053	2.909106	(4.475122825121185, 5.027609012960728, 5.02760...
4	1.000000e-12	(0, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...	153	0.006857	3.151855	(4.475137648839068, 5.027623836684378, 5.02762...
5	1.000000e-15	(0, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...	186	0.008051	2.905897	(4.4751381069387985, 5.027624294784101, 5.0276...

3.1.2 Forest Management 500 States

Similarly as with the 20 state Forest Management problem, it was just as complex to set up the 500 state Forest Management problem for VI,PI and Q learning so I followed the same process as in the 20 state Forest Management problem

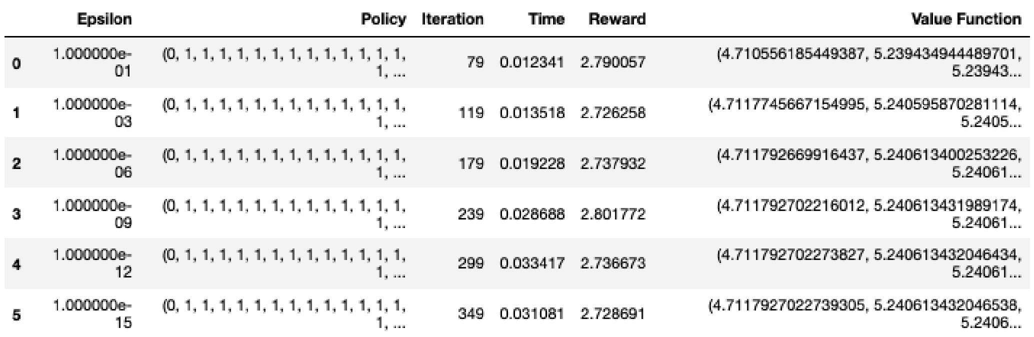
I ran the value iteration using the same parameter configurations as with the 20 state problem for easier comparison and the best evaluated expected reward for 1,000,000 value iterations was **value of 2.8** which was as expected lower than the 3.15 reward we had in the 20 state and also because we are overall increasing the probability of missing an optimal state when working with larger environment, thus bringing our overall expected value down.

For the policy iteration, I did the same thing again as with the 20 state set up, and interestingly the reward value was very close to what we got in the 20 state problem, at ***2.74*** but it took a lot more iterations to **46 iterations** to get this optimal value again unsurprising here because we are overall exploring a larger environment but still quite impressive that we were able to get an even better score in this case.

It also took more time overall for both Value and Policy Iterations to converge, at **~0.03 seconds and 0.15 seconds** respectively, with Policy iterations taking a lot longer here, but as we have discussed previously, policy iteration is less time efficient in larger environments compared to value iteration.

See table below for 6 epsilon parameter models for value iteration implementations.

Value Iteration Results Table



	Epsilon	Policy	Iteration	Time	Reward	Value Function
0	1.000000e-01	(0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, ...	79	0.012341	2.790057	(4.710556185449387, 5.239434944489701, 5.23943...
1	1.000000e-03	(0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, ...	119	0.013518	2.726258	(4.7117745667154995, 5.240595870281114, 5.2405...
2	1.000000e-06	(0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, ...	179	0.019228	2.737932	(4.711792669916437, 5.240613400253226, 5.24061...
3	1.000000e-09	(0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, ...	239	0.028688	2.801772	(4.711792702216012, 5.240613431989174, 5.24061...
4	1.000000e-12	(0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, ...	299	0.033417	2.736673	(4.711792702273827, 5.240613432046434, 5.24061...
5	1.000000e-15	(0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, ...	349	0.031081	2.728691	(4.7117927022739305, 5.240613432046538, 5.2406...

3.2 Q-learning

3.2.1 Forest Management 20 States

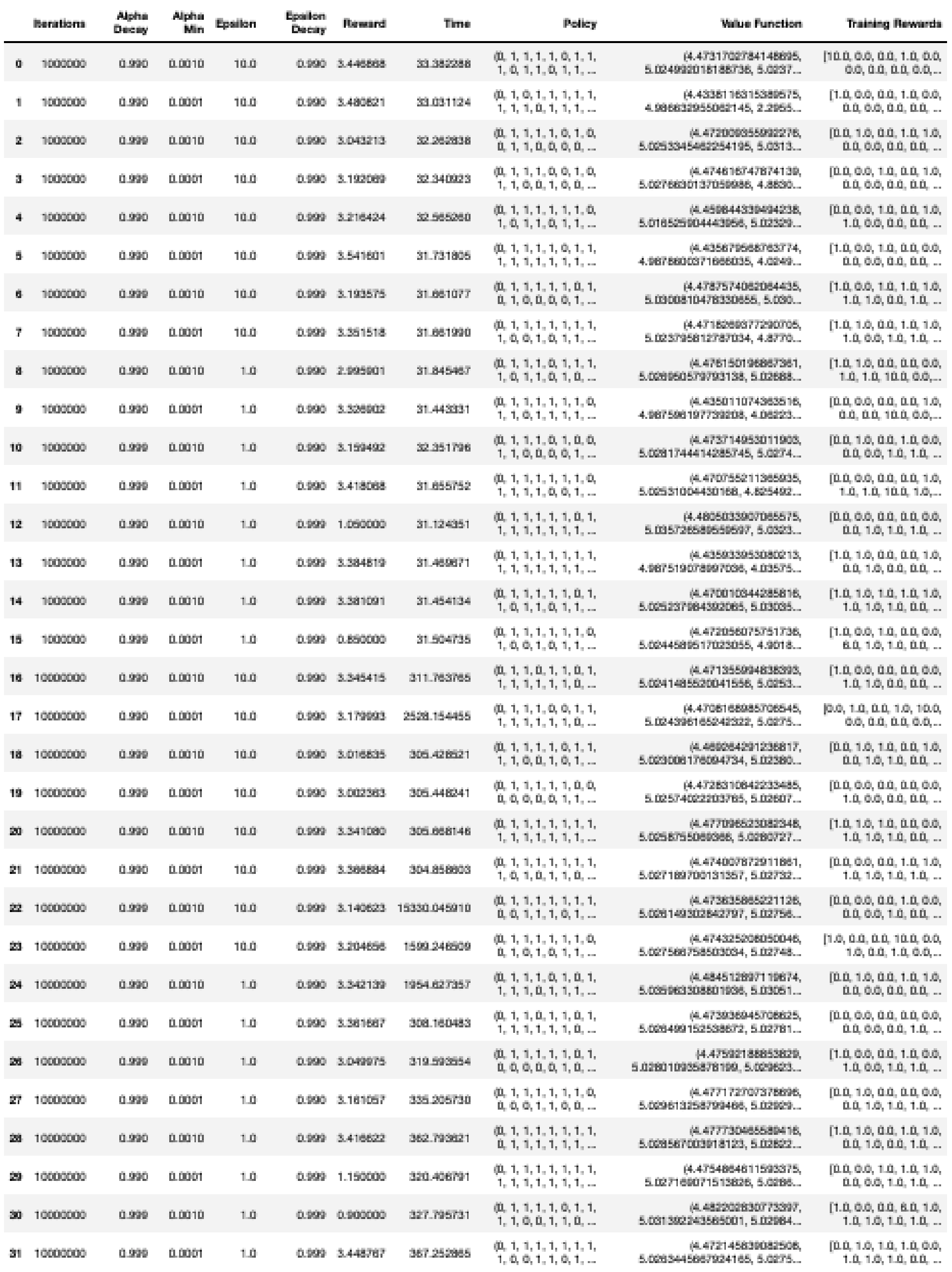
For the 20 state Forest Management problem, I set up the Q-Learning model with different parameter values for comparisons by adjusting the Epsilon, Learning rate and number of iterations. Looking at the results for each combination as shown in the table below, **the highest reward value of 3.54 was at epsilon = 10, at 1,000,000 iterations and learning rate of 0.0001**. This is a higher reward than what was achieved with the VI / PI models. It is also interesting to notice that there isn't a big difference in reward between one million to ten million iterations, which means with even relatively basic parameter values Q Learning is able to converge to an optimal reward value which is helpful when trying to simplify a model.

Timewise it took on average a lot longer for the Q-learning to to learn compared to the microseconds we saw with policy and value iteration, it took **31 seconds** for its best reward on 1,000,000 training episodes for a reward value that was less than 3 standard deviations away from the rewards with VI and PI.

In terms of exploration strategy I adjusted the learning rate, number of training episodes and epsilon value. As shown in the table below, we can see that the agent achieves the highest reward at **one million**

iterations, and at a **learning rate of 0.0001**. and for an **epsilon value of 10**. I think this combination worked best because the high decay rate and low learning rate balances each other out and the number of training examples is large enough for the agent to explore as much of the environment as possible.

Q-learning Results Table



	Iterations	Alpha Decay	Alpha Min	Epsilon	Epsilon Decay	Reward	Time	Policy	Value Function	Training Rewards
0	1000000	0.990	0.0010	10.0	0.990	3.446868	33.382288	(0, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 0, 1, 1, ...	(4.473170278414 8695, 5.024992018188736, 5.0237...	[10.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0,...
1	1000000	0.990	0.0001	10.0	0.990	3.480821	33.031124	(0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, ...	(4.433811631538 9575, 4.986632955062145, 2.2955...	[1.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, ...
2	1000000	0.999	0.0010	10.0	0.990	3.043213	32.262838	(0, 1, 1, 1, 1, 0, 1, 0, 0, 1, 1, 0, 0, 0, 0, ...	(4.472009355992276, 5.025334546225 4195, 5.0313...	[0.0, 1.0, 0.0, 1.0, 1.0, 0.0, 0.0, 0.0, 0.0, ...
3	1000000	0.999	0.0001	10.0	0.990	3.192069	32.340923	(0, 1, 1, 1, 0, 0, 1, 0, 1, 1, 0, 0, 1, 0, 0, ...	(4.474616747874139, 5.027663013705 9986, 4.8830...	[0.0, 0.0, 1.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, ...
4	1000000	0.990	0.0010	10.0	0.999	3.216424	32.565260	(0, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1, ...	(4.459844339494238, 5.016525904443956, 5.02329...	[0.0, 0.0, 1.0, 0.0, 1.0, 1.0, 0.0, 0.0, 0.0, ...
5	1000000	0.990	0.0001	10.0	0.999	3.541601	31.731805	(0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, ...	(4.435679568763774, 4.987880037166 6035, 4.0249...	[1.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, ...
6	1000000	0.999	0.0010	10.0	0.999	3.193575	31.661077	(0, 1, 1, 1, 1, 1, 0, 1, 0, 1, 0, 0, 0, 0, 1, ...	(4.478757406206 4435, 5.030081047833 0655, 5.030...	[1.0, 0.0, 1.0, 1.0, 1.0, 1.0, 1.0, 0.0, 1.0, ...
7	1000000	0.999	0.0001	10.0	0.999	3.351518	31.661990	(0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 0, 1, 1, ...	(4.471826937729 0705, 5.023795812787034, 4.8770...	[1.0, 1.0, 0.0, 1.0, 1.0, 1.0, 0.0, 1.0, 1.0, ...
8	1000000	0.990	0.0010	1.0	0.990	2.995901	31.845467	(0, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 0, 1, 0, ...	(4.476150196867361, 5.026950579793138, 5.02688...	[1.0, 1.0, 0.0, 0.0, 0.0, 1.0, 1.0, 10.0, 0.0,...
9	1000000	0.990	0.0001	1.0	0.990	3.326902	31.443331	(0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, ...	(4.435011074363516, 4.987596197739208, 4.06223...	[0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 10.0, 0.0,...
10	1000000	0.999	0.0010	1.0	0.990	3.159492	32.351796	(0, 1, 1, 1, 0, 1, 0, 0, 1, 1, 0, 0, 0, 0, 1, ...	(4.473714953011903, 5.028174441428 5745, 5.0274...	[0.0, 1.0, 0.0, 1.0, 0.0, 0.0, 0.0, 1.0, 1.0, ...
11	1000000	0.999	0.0001	1.0	0.990	3.418068	31.655752	(0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 0, 1, ...	(4.470755211365935, 5.02531004430168, 4.825492...	[0.0, 0.0, 0.0, 0.0, 1.0, 1.0, 1.0, 10.0, 1.0,...
12	1000000	0.990	0.0010	1.0	0.999	1.050000	31.124351	(0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, ...	(4.480503390706 5575, 5.035726589559597, 5.0323...	[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 1.0, 1.0, ...
13	1000000	0.990	0.0001	1.0	0.999	3.384819	31.469671	(0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, ...	(4.435933953080213, 4.987519078997036, 4.03575...	[1.0, 1.0, 0.0, 0.0, 1.0, 0.0, 1.0, 0.0, 0.0, ...
14	1000000	0.999	0.0010	1.0	0.999	3.381091	31.454134	(0, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 0, 1, 1, ...	(4.470010344285816, 5.025237984392065, 5.03035...	[1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 0.0, ...
15	1000000	0.999	0.0001	1.0	0.999	0.850000	31.504735	(0, 1, 1, 1, 1, 1, 1, 0, 1, 0, 0, 1, 0, 1, 1, ...	(4.472056075751736, 5.024458951702 3055, 4.9018...	[1.0, 0.0, 1.0, 0.0, 0.0, 6.0, 1.0, 1.0, 0.0, ...
16	10000000	0.990	0.0010	10.0	0.990	3.345415	311.763765	(0, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, ...	(4.471355994838393, 5.024148552004 1556, 5.0253...	[1.0, 0.0, 0.0, 0.0, 0.0, 1.0, 1.0, 0.0, 0.0, ...
17	10000000	0.990	0.0001	10.0	0.990	3.179993	2528.154455	(0, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, ...	(4.470816898570 6545, 5.024396165242322, 5.0275...	[0.0, 1.0, 0.0, 1.0, 10.0, 0.0, 0.0, 0.0, 0.0,...
18	10000000	0.999	0.0010	10.0	0.990	3.016835	305.428521	(0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 0, 1, 0, 1, ...	(4.469264291236817, 5.023008176094734, 5.02380...	[0.0, 1.0, 1.0, 0.0, 1.0, 0.0, 1.0, 1.0, 0.0, ...
19	10000000	0.999	0.0001	10.0	0.990	3.002363	305.446241	(0, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, ...	(4.472831084223 3485, 5.025740222037 65, 5.02607...	[0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, ...
20	10000000	0.990	0.0010	10.0	0.999	3.341080	305.668146	(0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, ...	(4.477096523082348, 5.025875506 9366, 5.0280727...	[1.0, 1.0, 1.0, 0.0, 0.0, 1.0, 1.0, 1.0, 0.0, ...
21	10000000	0.990	0.0001	10.0	0.999	3.366884	304.858803	(0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 0, ...	(4.474007872911861, 5.027189700131357, 5.02732...	[0.0, 0.0, 0.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, ...
22	10000000	0.999	0.0010	10.0	0.999	3.140623	15330.045910	(0, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 0, 1, ...	(4.473635865221126, 5.028149302842797, 5.02756...	[0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 1.0, 0.0, ...
23	10000000	0.999	0.0001	10.0	0.999	3.204656	1599.246509	(0, 1, 1, 1, 1, 1, 1, 0, 0, 1, 0, 1, 0, 1, 1, ...	(4.474325208050046, 5.027566758503034, 5.02748...	[1.0, 0.0, 0.0, 10.0, 0.0, 1.0, 0.0, 1.0, 0.0,...
24	10000000	0.990	0.0010	1.0	0.990	3.342139	1954.627357	(0, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, ...	(4.484512897119674, 5.035963308801936, 5.03051...	[0.0, 1.0, 0.0, 1.0, 1.0, 0.0, 0.0, 0.0, 0.0, ...
25	10000000	0.990	0.0001	1.0	0.990	3.381687	308.160483	(0, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, ...	(4.473936945708625, 5.026499152538672, 5.02781...	[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, ...
26	10000000	0.999	0.0010	1.0	0.990	3.049975	319.593554	(0, 1, 1, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, ...	(4.47592188853829, 5.028010935878199, 5.029623...	[1.0, 0.0, 0.0, 1.0, 0.0, 1.0, 0.0, 1.0, 1.0, ...
27	10000000	0.999	0.0001	1.0	0.990	3.161057	335.205730	(0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 1, 1, 0, 0, ...	(4.477172707378696, 5.029613258799466, 5.02929...	[0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 1.0, 1.0, 1.0, ...
28	10000000	0.990	0.0010	1.0	0.999	3.416622	362.793621	(0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, ...	(4.477730465589416, 5.028567003918123, 5.02822...	[1.0, 1.0, 0.0, 1.0, 1.0, 0.0, 1.0, 0.0, 1.0, ...
29	10000000	0.990	0.0001	1.0	0.999	1.150000	320.406791	(0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, ...	(4.475486461159 3375, 5.027169071513826, 5.0286...	[0.0, 0.0, 1.0, 1.0, 1.0, 0.0, 0.0, 1.0, 1.0, ...
30	10000000	0.999	0.0010	1.0	0.999	0.900000	327.795731	(0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 0, 1, 1, 0, ...	(4.482202830773397, 5.031392243565001, 5.02984...	[1.0, 0.0, 0.0, 6.0, 1.0, 1.0, 1.0, 1.0, 1.0, ...
31	10000000	0.999	0.0001	1.0	0.999	3.448767	367.252865	(0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 0, 1, ...	(4.472145839082508, 5.026344566792 4165, 5.0275...	[0.0, 1.0, 1.0, 1.0, 0.0, 1.0, 1.0, 1.0, 0.0, ...

3.2.2 Forest Management 500 States

Lastly for the 500 state Forest Management problem, I set up the Q-Learning with the same configurations as with the 20 states. The **highest reward value was 2.86 was at epsilon = 10, at 10,000,000 iterations and learning rate of 0.0001**. This is a lower reward than at the 20 state and also lower reward than what was achieved with the VI model. It is also interesting to notice that it took 10 million iterations to reach this higher reward compared to one million rewards in the 20 state, and with the reward being lower overall. Again a lot of it makes sense since we are exploring a larger environment.

Timewise it took on average a lot longer at 4**35 seconds** for its best reward on 10,000,000 training episodes for a reward value that was lower than at 20 states and for both VI iterations. But taking a longer time for ten millions iterations also makes sense.

In terms of exploration strategy I did the same thing as with the 20 state by adjusting the learning rate, number of training episodes and epsilon value. As shown in the table below, we can see that the agent achieves the highest reward at **ten million iterations**, and at a **learning rate of 0.0001**. and for an **epsilon value of 10**. I think this combination worked best because the high decay rate and low learning rate balances each other out and the number of training examples is large enough for the agent to explore as much of the environment as possible.

4 CONCLUSIONS

After evaluating Value Iterations, Policy Iterations and Reinforcement learning on two different types of problems of different sizes each, I came to the following conclusions about when and how to use each model..

Iterations In terms of iterations, we've seen that Policy Iterations is the way to go both on grid and non-grid MDP problems for a still somewhat optimal reward value, although not the best.

Timewise, Value iterations is the winner as its runtime does not increase as much as the complexity and size of the environment grows compared to policy iterations and Q Learning.

Complexity In terms of complexity, Q-learning although it seems to be more complex is the most straightforward because we do not need as much background knowledge about the environment's states and rewards as we do with Value Iteration and Policy iterations in order to set it up. It also easier to achieve higher rewards value with Q Learning compared to with Value Iterations and policy iterations.

Reward Overall Q-learning consistently produced the highest reward values on all the problems with both small and large environments, despite how expensive it was sometimes.

5 REFERENCES

1. MDP Forest Management Example https://pymdptoolbox.readthedocs.io/en/latest/api/mdp.html#mdptoolbox.mdp.ValueIteration

2. Value Iteration to solve OpenAI Gym's FrozenLake https://towardsdatascience.com/value-iteration-to-solve-openai-gyms-frozenlake-6c5e7bf0a64d
3. Frozen Lake: Beginners Guide To Reinforcement Learning With OpenAI Gym https://analyticsindiamag.com/openai-gym-frozen-lake-beginners-guide-reinforcement-learning/
4. Tiny Forest management problem with R using Reinforcement learning https://learning.oreilly.com/library/view/hands-on-reinforcement-learning/9781789616712/c03dc216-f78b-4c5a-9ba4-d8ec7b9c191f.xhtml
5.