Kellen O'Connor

OMSCS-7641

Assignment 4

Markov Decision Processes

Summary:

In this paper we implement and examine two different Markov decision processes, namely forest management and a frozen lake grid world experiment. We want to evaluate MDP using policy iteration, value iteration, and Q-Learning algorithms, and then optimize these algorithms in the respective MDP problem. Interesting findings or surprises are explored and analyzed.

The MDP Environments:

Forest Management

The Forest Management problem is my chosen non grid world environment that has a forest that grows to a predetermined maximum. At each step a choice needs to be made to either cut or wait. If the agent decides to cut then it gets a reward of 1 and the forest state returns to 0 where it can't be cut the next turn. At each turn if the agent decides to wait, the environment will continue on to the next state with probability p, being the likelihood of a forestfire. There is a reward given at the end for the environment getting to it's final state and the agent deciding to cut, and there is a different reward given for a decision to wait at this time, with more points given to a decision to wait.

Frozen Lake

Our frozen lake simulation is our grid world environment where an agent tries to from the top left of the environment to the bottom right. This simulation has hole tiles which end the simulation with no reward, regular tiles that continue the simulation and provide no reward, and then a reward point of 1 for arriving at the finishing tile in the bottom right of the grid world environment. The actions the agent can take while traversing the grid world include moving up, left, down, and right. I have used two sizes of the grid world to observe the effects of using our different algorithms policy iteration, value iteration and Q-Learning.

Forest Management (25)

In our forest management simulation the discount rate must be a design decision instead of a hyperparameter for our provided algorithms. This is because throughout the simulation there is more than one reward unlike Frozen lake's terminal grid space at the bottom right. As stated previously the agent in this simulation accrues rewards as the game progresses with each turn via cut or wait decisions. In changing the discount rate, we also deal with the different possible environments that have varying optimal policies. This simulation has inconsistent run times, so in order to capture information about performance we run the agent from each state 1000 times and then get the average reward given a particular state. I have chosen to give r1 = 10 for keeping the forest in its final state, and then r2 = 6 for cutting the forest in its final state. The value of P is 0.1.

Value Iteration (FM 25)

For value iteration I've chosen to use differing epsilon values and a constant for the discount value. Since there are a bunch of rewards depending on the simulation state for my 25 state environment smaller epsilon values seemed to have diminished impact on the policy as all were small enough to get the optimum policy with little distinction between values of epsilon. I've tried 1e-1, 1e-3, 1e-6, 1e-9, 1e-12, and 1e-15 as shown in the table below.

	Epsilon	Policy	Iteration	Time	Reward	Value Function
0	1.000000e-01	(0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, ...	33	0.001995	2.503315	(4.328504830081768, 4.881518644971712, 4.88151...
1	1.000000e-03	(0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, ...	55	0.001995	2.552850	(4.460720290173723, 5.013211594807497, 5.01321...
2	1.000000e-06	(0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, ...	87	0.003989	2.550820	(4.474643139169861, 5.027129333047953, 5.02712...
3	1.000000e-09	(0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, ...	120	0.004986	2.461267	(4.475122825121185, 5.027609012960728, 5.02760...
4	1.000000e-12	(0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, ...	153	0.006965	2.537514	(4.475137648839068, 5.027623836684378, 5.02762...
5	1.000000e-15	(0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, ...	186	0.006981	2.521253	(4.4751381069387985, 5.027624294784101, 5.0276...

We can see in the reward column that despite changing the value of epsilon we see very little change in reward with no indication that increasing epsilon has a predictable effect on the reward. Furthermore for larger values of epsilon we see increased iteration counts and therefore longer times. Because of the negligible impact on reward and the simplicity of the simulation value iteration does a decent job at returning an optimal reward at any epsilon value. It is likely that we would select a smaller value for epsilon to decrease runtime.

Policy Iteration (FM 25)

For this assignment I've used python's mdptoolbox's implementation of the forest management problem so I was unable to test for varying levels of epsilon. Our implementation of policy iteration for forest management gave the same mean rewarded policy with all of the value iterations, doing the 14 iterations in 0.04 seconds. From this output we can observe that policy iteration is noticeably slower to any of the Value Iteration runs we experimented with above.

Q-Learning (FM 25)

Q-Learning typically uses an agent to explore an environment space, making this sort of sequentially chained simulation a little bit strange for Q-Learning. To better optimize our Q-Learning implementation I have evaluated 5 different hyper-parameters to see how or if it's able to converge to an optimum state. This implementation explore number of iterations, learning rate, learning-rate decay, epsilon and epsilon decay. The results of this exploration are in the table on the right.

As shown in the mentioned table, the policies produce similar results although there are a few standout observations. Simulation runs with small AlphaMin (0.0001) and alpha decay (0.990) seem to return the best reward. Also none of the simulation runs with number of iterations 10000000 do best, suggesting that Q-Learning is able to converge with only 100000 iterations instead of 10000000. All in all, Q-Learning takes much longer than either value iteration or policy iteration for the 25 state forest management simulation, frequently taking at least ten minutes or more.

Iterations	Alpha Decay	Alpha Min	Epsilon	Epsilon Decay	Reward	Time
1000000	0.9945	0.00055	5.5	0.9945	2.636017	32.564421
10000000	0.9945	0.00055	5.5	0.9945	2.511882	317.588549

	Iterations	Alpha Decay	Alpha Min	Epsilon	Epsilon Decay	Reward	Time
0	1000000	0.990	0.0010	10.0	0.990	2.721332	36.888807
1	1000000	0.990	0.0001	10.0	0.990	2.932481	32.110655
2	1000000	0.999	0.0010	10.0	0.990	2.621837	32.343206
3	1000000	0.999	0.0001	10.0	0.990	2.569055	32.326625
4	1000000	0.990	0.0010	10.0	0.999	2.913671	32.391937
5	1000000	0.990	0.0001	10.0	0.999	3.036907	32.314988
6	1000000	0.999	0.0010	10.0	0.999	2.604180	32.366592
7	1000000	0.999	0.0001	10.0	0.999	2.696395	32.181986
8	1000000	0.990	0.0010	1.0	0.990	2.689540	32.320821
9	1000000	0.990	0.0001	1.0	0.990	3.001139	32.148795
10	1000000	0.999	0.0010	1.0	0.990	2.570384	32.373944
11	1000000	0.999	0.0001	1.0	0.990	2.542357	32.138888
12	1000000	0.990	0.0010	1.0	0.999	2.831567	32.247681
13	1000000	0.990	0.0001	1.0	0.999	2.835339	32.322815
14	1000000	0.999	0.0010	1.0	0.999	2.730079	32.308396
15	1000000	0.999	0.0001	1.0	0.999	0.880000	32.244605
16	10000000	0.990	0.0010	10.0	0.990	2.860680	331.675365
17	10000000	0.990	0.0001	10.0	0.990	2.850239	320.504785
18	10000000	0.999	0.0010	10.0	0.990	2.691968	316.834189
19	10000000	0.999	0.0001	10.0	0.990	2.542538	315.642150
20	10000000	0.990	0.0010	10.0	0.999	2.818113	316.436175
21	10000000	0.990	0.0001	10.0	0.999	1.080000	315.705816
22	10000000	0.999	0.0010	10.0	0.999	0.920000	314.506162
23	10000000	0.999	0.0001	10.0	0.999	2.651629	316.717407
24	10000000	0.990	0.0010	1.0	0.990	2.815815	316.208655
25	10000000	0.990	0.0001	1.0	0.990	2.936016	316.863702
26	10000000	0.999	0.0010	1.0	0.990	2.546502	316.499335
27	10000000	0.999	0.0001	1.0	0.990	2.444163	317.609994
28	10000000	0.990	0.0010	1.0	0.999	2.818473	316.001820
29	10000000	0.990	0.0001	1.0	0.999	2.863827	317.374356
30	10000000	0.999	0.0010	1.0	0.999	2.652020	316.674359
31	10000000	0.999	0.0001	1.0	0.999	2.698134	316.162512

Generally speaking, across all 5 hyperparameters and between 10000000 and 100000 iteration runs it seems that although taking longer than value iteration or policy iteration, Q-Learning is able to reasonably converge regardless of the hyperparameter values that I have tested.

Forest Management (FM 600)

Wanting to examine the difference between small and large non-grid world problems this second implementation of forest management contains 600 states versus our initial 25. In this implementation of the simulation I've decided to use r1 = 80, r2 = 10, and p = 0.01. I've swapped the discount value from 0.9 to 0.99 because of the length of this simulation and a desire for it to not be disrupted prematurely.

Value Iteration (FM 600)

Just as before with my 25 state forest management simulation I've opted to use the same epsilon values 1e-1, 1e-3, 1e-6, 1e-9, 1e-12, and 1e-15. The values of testing against these levels of epsilon are shown below in the 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	0.009971	2.437386	(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	0.014960	2.446311	(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	0.025902	2.427971	(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	0.030889	2.469375	(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	0.039895	2.436066	(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	0.054853	2.438026	(4.7117927022739305, 5.240613432046538, 5.2406...

From this output we can tell that just like before, simulation reward for differing levels of epsilon are pretty close to one another. Epsilon 1e-1 can capture a decent reward value but also take remarkably less time compared to its larger values of epsilon, despite being similar in reward. One observation that I think we can glean from this and our 25 state simulation is that value iteration is luckily fairly linear against number of states (25 vs 600, etc). This is good for scaling up value iteration for larger problems.

Policy Iteration (FM 600)

As stated above for this assignment I have been using mdptoolbox's implementation of the forest management problem so I was unable to test for varying levels of epsilon. Our implementation of policy iteration for forest management gave the same mean rewarded policy with all of the value iterations, doing the 46 iterations in 0.43 seconds. Similar to what we observed from the 25 state simulation, policy iteration takes longer than value iteration to converge. One noticeably promising find is that like the scaling of value iteration from 25 to 600 state, the scaling of policy iteration from 25 states to 600 also appears to be fairly linear in terms of runtime. This will benefit projects or simulations that require many more iterations than forest management.

Q-Learning (FM 600)

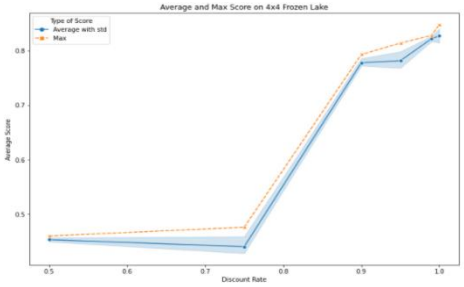
Implemented against this run of the simulation in the same way we implemented it for forest management with 25 states, we use evaluate Q-learning against 5 hyperparameters. Hyperparameters explored include number of iterations, learning rate, learning-rate decay, epsilon and epsilon decay. You can see the values from this exploration in the table on the right.

Just like what we observed in the 25 state simulation there doesn't seem to be an reliable relationship between reward and the different hyperparameter values we explore here meaning and it seems like Q-Learning is able to converge at an optimal with 100000 iterations instead of needing to use all 10000000 iterations. However when we look at the averages across # of iterations we do see a slight improvement, probably indicating a slight increase in reward from additional iterations.

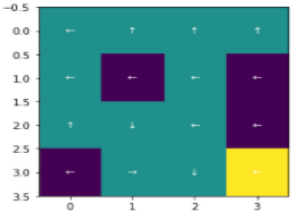
Iterations	Alpha Decay	Alpha Min	Epsilon	Epsilon Decay	Reward	Time
1000000	0.9945	0.00055	5.5	0.9945	2.118720	46.191236
10000000	0.9945	0.00055	5.5	0.9945	2.468694	425.352037

	Iterations	Alpha Decay	Alpha Min	Epsilon	Epsilon Decay	Reward	Time
0	1000000	0.990	0.0010	10.0	0.990	2.314850	44.333165
1	1000000	0.990	0.0001	10.0	0.990	2.273474	44.469392
2	1000000	0.999	0.0010	10.0	0.990	2.262396	41.611148
3	1000000	0.999	0.0001	10.0	0.990	2.335135	46.105560
4	1000000	0.990	0.0010	10.0	0.999	2.331942	48.478974
5	1000000	0.990	0.0001	10.0	0.999	2.380295	49.075365
6	1000000	0.999	0.0010	10.0	0.999	2.325847	46.995536
7	1000000	0.999	0.0001	10.0	0.999	2.305543	48.149884
8	1000000	0.990	0.0010	1.0	0.990	2.268132	44.884726
9	1000000	0.990	0.0001	1.0	0.990	0.793333	44.992038
10	1000000	0.999	0.0010	1.0	0.990	0.810000	44.461492
11	1000000	0.999	0.0001	1.0	0.990	2.328302	45.414015
12	1000000	0.990	0.0010	1.0	0.999	2.239858	43.244794
13	1000000	0.990	0.0001	1.0	0.999	2.384879	49.406977
14	1000000	0.999	0.0010	1.0	0.999	2.221562	47.984002
15	1000000	0.999	0.0001	1.0	0.999	2.323969	49.452709
16	10000000	0.990	0.0010	10.0	0.990	2.395558	438.899460
17	10000000	0.990	0.0001	10.0	0.990	2.529896	405.098862
18	10000000	0.999	0.0010	10.0	0.990	2.441515	462.404932
19	10000000	0.999	0.0001	10.0	0.990	2.476812	447.818858
20	10000000	0.990	0.0010	10.0	0.999	2.469757	464.364363
21	10000000	0.990	0.0001	10.0	0.999	2.535612	470.487227
22	10000000	0.999	0.0010	10.0	0.999	2.442954	443.605574
23	10000000	0.999	0.0001	10.0	0.999	2.501121	421.663777
24	10000000	0.990	0.0010	1.0	0.990	2.442936	414.839873
25	10000000	0.990	0.0001	1.0	0.990	2.502709	404.318833
26	10000000	0.999	0.0010	1.0	0.990	2.493374	406.608002
27	10000000	0.999	0.0001	1.0	0.990	2.513437	403.201386
28	10000000	0.990	0.0010	1.0	0.999	2.424003	406.476201
29	10000000	0.990	0.0001	1.0	0.999	2.512902	403.888571
30	10000000	0.999	0.0010	1.0	0.999	2.349643	406.681973
31	10000000	0.999	0.0001	1.0	0.999	2.466878	405.304700

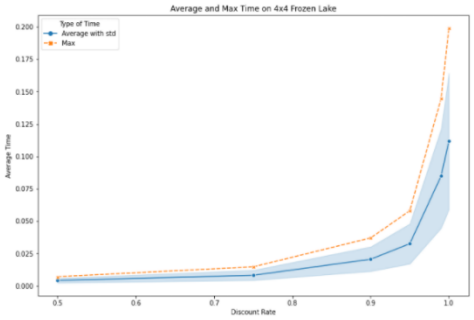
Value Iteration (Frozen Lake 4x4)

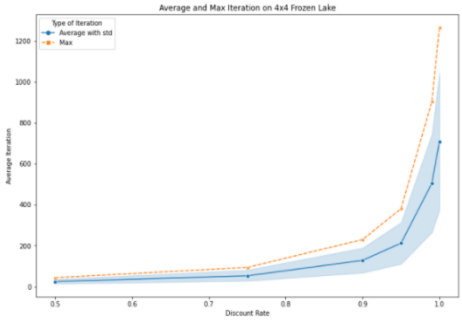


For my implementation of the frozen lake grid-world simulation I've opted to use different discount rates, effecting the final values of non-terminal grid locations and the epsilon value that is used to influence algorithm convergence. The discount rates used are 0.5, 0.75, 0.9, 0.95, 0.99, and 0.9999. For values of epsilon I have explored the same values used in the forest management exercise which were 1e-6, 1e-9, 1e-12, and 1e-15. To better analyze rewards I've ran 1000 sessions of the simulation and have taken averages from those 1000 runs.

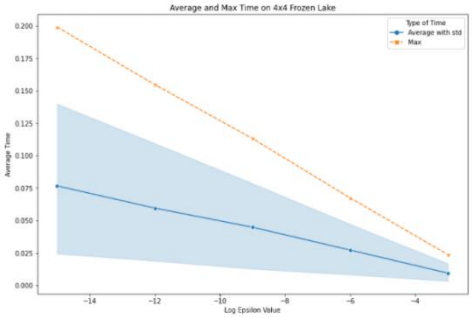


Something I observed immediately was that the discount rates have an extremely significant impact on simulation score. In particular at discount rate 0.75 there starts to be dramatic improvement in algorithm performance. On average any simulation run with a discount rate of 0.9 or higher has at least a 75% chance of arriving at the goal tile square in the bottom right of the gridworld.



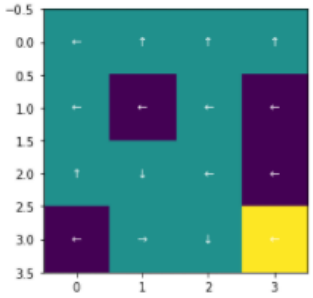


Based on the plots provided here it can be seen that the discount rate also impacts the time spent to create policies and iterations considerably. Because higher discount rates also generally impact reward achieved it makes sense to use a discount rate about 0.9 for this particular simulated environment.



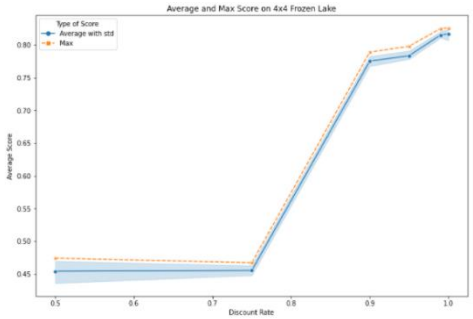
This final plot for value iteration highlights the effect of varying epsilon values on simulation runtime. In particular our best average scoring epsilon value 1e-6 has an average time of only a little over 0.025. Moreover it can be seen from this plot that as epsilon get's smaller there is a linearly increasing relationship with average runtime.

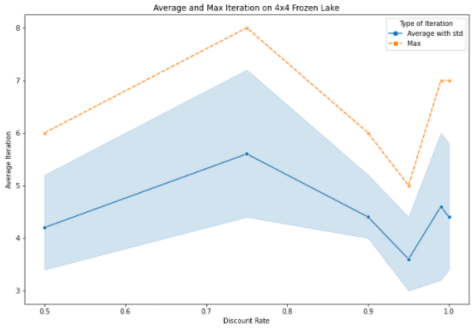
Policy Iteration (Frozen Lake 4x4)

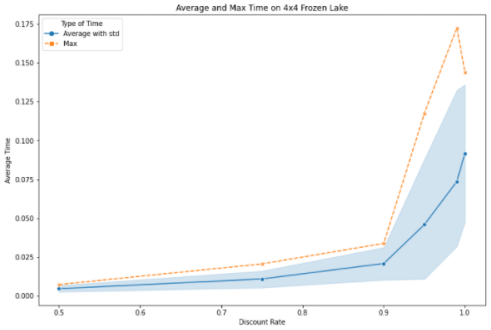


For my implementation of policy iteration against the 4 by 4 frozen lake environment I've used two hyperparameters, discount rate and epsilon. Giving it an initial run returns a policy extremely similar to the policy returned by value iteration. Because of this, this policy returns a similar reward level to what we observed in value iteration above.

Additionally, just like with value iteration we see a considerable emphasis on the importance of the value of discount rate on simulation score, as show in the plot to the left. Just like before any discount rate above 0.9 allows the simulation to consistently arrive at the terminal grid space and be awarded a point. Interestingly, the relationship between average iterations and average max time for the simulations seems to differ quite a bit from value iteration and doesn't present a clear relationship in the same way that value iteration did.



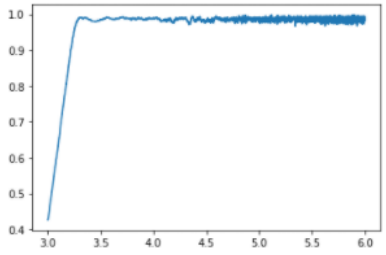


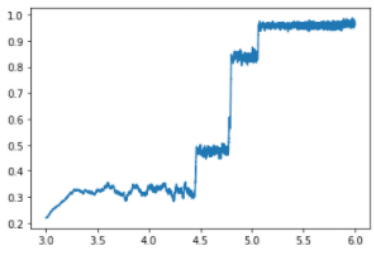


Another observation from the maximum time plot on the right is that policy iteration appears to be able to complete with fewer iterations than value iteration with for our discount rate of 0.9. Lastly, epsilon value does not appear to have any significant impact on average score or average iterations and similar to value iteration negatively impacts total runtime when smaller values of epsilon are used (1e-9, 1e-12, 1e-15, etc).

Q-Learning (Frozen Lake 4x4)

For my implementation of Q-Learning against frozen lake I've decided to explore hyperparameters discount rate and learning rate as others seemed to have a less significant effect on simulation performance. Plotted below are rolling means for rewards in the training phase to visualize convergence for two values of learning rate 0.1 and 0.01. For comprehension the x axis is log normalized so 6 represents 1,000,000.



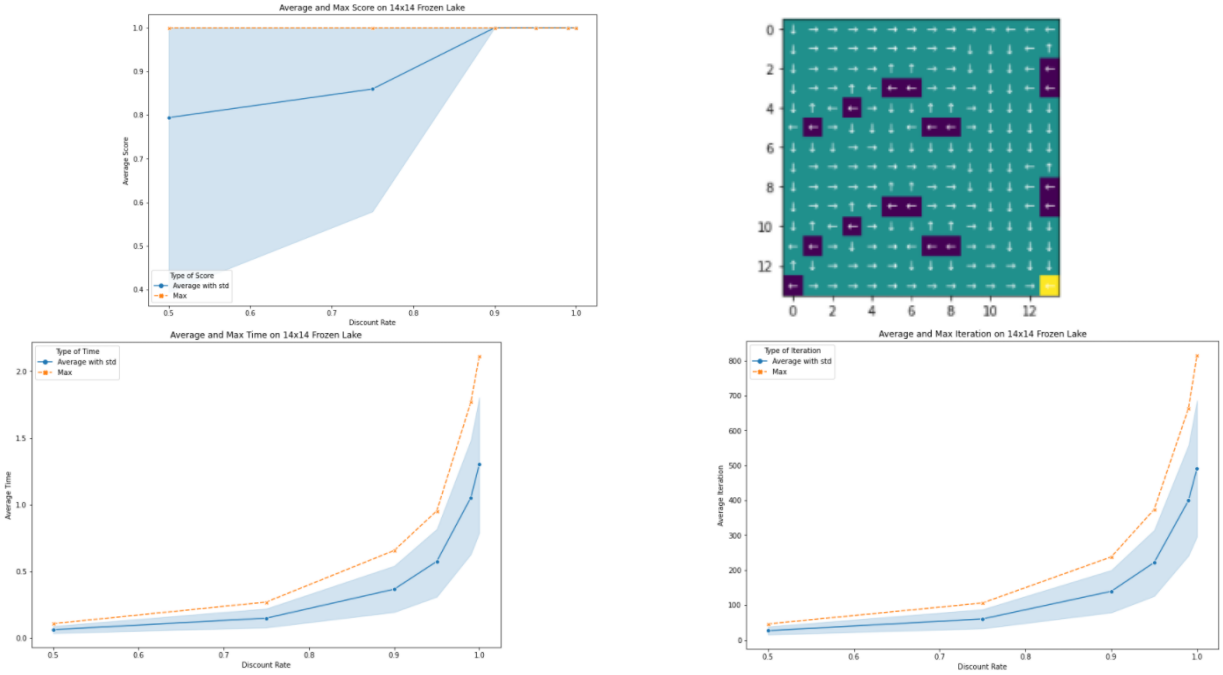


	Discount Rate	Training Episodes	Learning Rate	Decay Rate	Reward	Time Spent
0	0.9999	10000.0	0.10	0.00100	1.000	7.036631
1	0.9999	10000.0	0.10	0.00001	1.000	2.821746
2	0.9999	10000.0	0.01	0.00100	0.197	2.094916
3	0.9999	10000.0	0.01	0.00001	1.000	2.822633
4	0.9999	100000.0	0.10	0.00100	1.000	66.161246
5	0.9999	100000.0	0.10	0.00001	1.000	43.966147
6	0.9999	100000.0	0.01	0.00100	1.000	60.055866
7	0.9999	100000.0	0.01	0.00001	1.000	42.708599
8	0.9999	1000000.0	0.10	0.00100	1.000	628.120350
9	0.9999	1000000.0	0.10	0.00001	1.000	595.132914
10	0.9999	1000000.0	0.01	0.00100	1.000	965.181906
11	0.9999	1000000.0	0.01	0.00001	1.000	526.222648

From these plots we can tell that with the learning rate of 0.1 Q-Learning is able to very quickly find an optimal decision policy for the 4 by 4 grid world. For a learning rate of 0.01 it takes some time to eventually start to converge on the optimal but does so. When we look at the table of Q-Learning evaluated against our different hyperparameters we see that for almost every level of every hyperparameter Q-Learning is able to settle on the optimal. At first I was surprised by this but suppose it's just due to the four by four grid world environment being fairly simple and Q-Learning is able to converge upon a policy that almost always gets it to the reward.

Value Iteration (Frozen Lake 14x14)

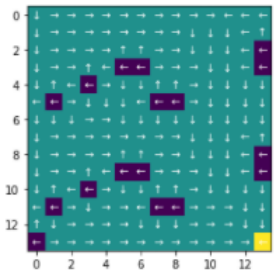
Just as before for our 4 by 4 frozen lake environment I implement value iteration for the simulation using discount rates 0.5, 0.75, 0.9, 0.95, 0.99, and 0.9999. For values of epsilon I have explored the same values used in the forest management exercise which were 1e-6, 1e-9, 1e-12, and 1e-15. To better analyze rewards I've ran 1000 sessions of the simulation and have taken averages from those 1000 runs. Examining the plot below shows remarkably better performance in our 14x14 grid world versus the 4x4 grid world. Plotting out it's policy I am assuming that the simulation is able to navigate the grid and avoid holes in the grid world much easier than in the 4x4 because of the sheer number of possible routes to the bottom route.

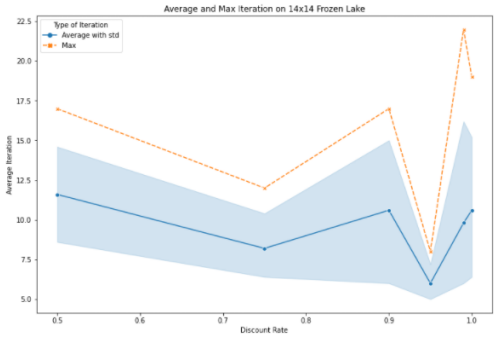


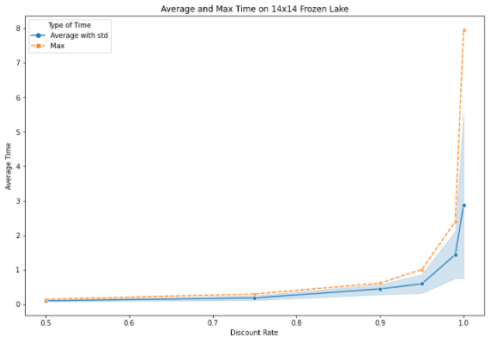
When we plot the average iterations and the average times for the 14 by 14 grid world we get the plots provided above. Despite a similar pattern to our 4 by 4 grid world, the values on the y axis are considerably different. If we look again at the output for the 4 by 4 grid world we can see that average time for a 0.95 discount rate was 0.02 seconds whereas for the 14 by 14 grid world the average time for 0.95 discount rate is over half a second (>0.5). This is interesting because this relationship doesn't hold up for max iteration where the 4 by 4 environment and the 14 by 14 environment both hold an average max iteration of 500.

Policy Iteration (Frozen Lake 14x14)

For my implementation of policy iteration against the same 14 by 14 grid world, policy iteration was actually entirely capable of achieving an average score of 1 across all values of epsilon. I will avoid plotting the horizontal line plot in the interest of space. Looking at the policy plot it seems most likely that PI is able to learn the path of starting at the top right, navigating to the right and then downwards to avoid the clumps of holes to the center, bottom and left and settles on a policy to achieve this type of navigation.



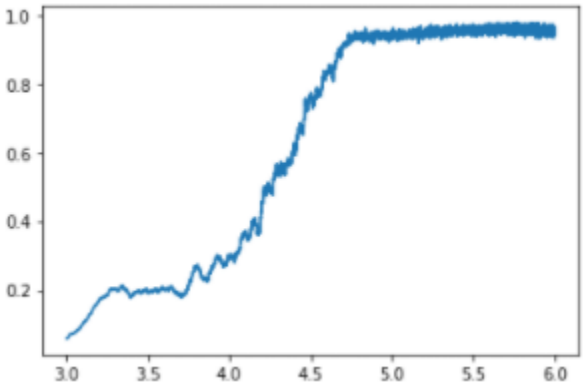


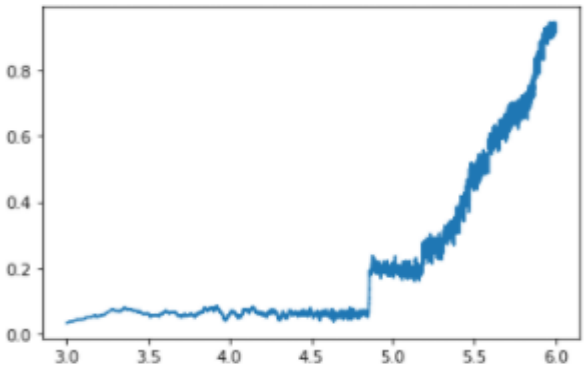


An interesting observation is the increase in time for the 14 by 14 grid world over the 4 by 4 grid world. The 4 by 4 simulation took roughly half a second via policy iteration whereas the 14 by 14 simulation took about 3 seconds. I really thought we would see a much larger increase in runtime between the two gird world environments as the dimensions of the space enlarged quite a bit, but we did not. We do see an increase in the average iterations for the 14 by 14 simulation over the 4 by 4 simulation but this makes intuitive sense as the environment itself is larger.

Q-Learning (Frozen Lake 14x14)

For my implementation of Q-Learning against this 14 by 14 frozen lake environment I've explored hyperparameters discount rate and learning rate as others seemed to have a less significant effect on simulation performance. Plotted below are rolling means for rewards in the training phase to visualize convergence for two values of learning rate 0.1 and 0.01. For comprehension the x axis is log normalized so 6 represents 1,000,000.





From these plots we can observe that with learning rate 0.1 Q-Learning takes some time to converge on the optimal solution and then oscilates a little towards the top of the plot near the 1.0 reward. In the 0.01 plot however we see that Q-Learning takes a considerable amount of time to even begin the convergence towards the optimal, only barely getting there after almost a million. We are not able to see how it behaves before then but the relationship between these two plots does show us that a larger learning rate is probably needed to reliably converge towards the optimal solution.

	Discount Rate	Training Episodes	Learning Rate	Decay Rate	Reward	Time Spent
0	0.9999	10000.0	0.10	0.00100	0.166	9.185713
1	0.9999	10000.0	0.10	0.00001	0.635	6.746721
2	0.9999	10000.0	0.01	0.00100	0.040	12.012376
3	0.9999	10000.0	0.01	0.00001	0.481	6.309835
4	0.9999	100000.0	0.10	0.00100	0.994	349.115830
5	0.9999	100000.0	0.10	0.00001	1.000	120.889552
6	0.9999	100000.0	0.01	0.00100	0.278	108.462017
7	0.9999	100000.0	0.01	0.00001	0.805	109.620789
8	0.9999	1000000.0	0.10	0.00100	0.998	3674.318641
9	0.9999	1000000.0	0.10	0.00001	1.000	2147.411797
10	0.9999	1000000.0	0.01	0.00100	0.976	2784.848195
11	0.9999	1000000.0	0.01	0.00001	0.983	3829.177074

When we show the table with accumulated data from our simulation runs we can better observe how the different hyperparameters influenced simulation performance. From the table we can see that the learning rate probably has the largest effect on performance followed by the number of training episodes, and then the decay rate. The strongest performing simulations had a learning rate of 0.1 a decay rate of 0.00001, and at least 100000 training episodes. All of this suggests that the solution takes some time to arrive at and that the model must either have

enough episodes to converge, or a high enough learning rate to converge faster, or both. All of the simulations with more training episodes took longer to converge which is intuitive so it is likely best to choose 100000 training episodes instead of 10000000 and use a higher learning rate.

Conclusion

We have implemented 3 algorithms (VI, PI, and Q-Learning) against two different simulation environments, forest management and frozen lake. The two model based algorithm's seemed to produce the best policy over the Q-Learning method. I believe this is the case because PI and VI are model based whereas Q-Learning needs to progressively interact with the simulation environment to properly converge. In our 14 by 14 frozen lake grid world experiment we can see signs of this where Q-Learning takes considerable time to converge when it doesn't have a high enough learning rate. Between the grid world and forest management environments it seemed as if all algorithms and Q-Learning struggled a bit more than in frozen lake and I think this was because of the sequential or chained state nature of the experiment that differed from Frozen Lake.

References

Sai Sasank, "Frozen-Lake as a Markov Decision Process," Medium, December 13, 2020, https://medium.com/swlh/frozen-lake-as-a-markov-decision-process-1692815ecfd1;

OpenAI, "Gym: A Toolkit for Developing and Comparing Reinforcement Learning Algorithms," accessed April 25, 2021, https://gym.openai.com;

Pieter Abbeel, "Markov Decision Processes Value Iteration," n.d., 6.