Markov Decision Processes

Jun Wang
CS7641 – Assignment 4
Jwang3316@gatech.edu

Abstract

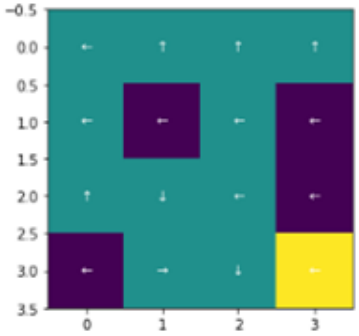
In MDP analysis, I come up with two interesting MDP problems. Frozen Lake and Forest Management. Why they are interesting is explained in the next section. I used 2 planning algorithms, Value Iteration as well as Policy Iteration, and also I pick Q-Learning as my favorite reinforcement learning algorithm and use it to solve the two MDPs. The results, the performance, iterations/time to converge and the exploration strategies in different environments are discussed and analyzed.

1. Discription of MDPs & Why interesting

Due to the requirement of this assignment, I choose two MDPs problems with different numbers of states. First, in Frozen Lake grid-world environment, I set two cases for comparison (one is 16 states, one is 256 states). Second, the Forest Management environment, the non-grid world environment with 'small' (20) and 'large' (500) number of states are discussed.

1.1 Frozen Lake

Here in Frozen Lake [1], the rule is very simple: we attempt to move from the top left grid to the bottom right grid. There are **holes** in the environment. If we move on the holes, the action ends with a 0 reward. If we successfully arrive at the lower right grid, we get 1 reward.



In the map, I designed the holes and destination in different colors: holes are purple (dangerous) and the destination with reward is yellow.

MDP tuples ((S,A,P,R,γ)):

Actions: UP/DOWN/LEFT/RIGHT

Probability: since the environment is stochastic, the next grid we move to depends on the probability. Each direction has the same probability: 0.33. For example, when the action is UP, the probability that we finally go UP is 0.33, which is same as we go LEFT and RIGHT.

I designed two different sizes for this MDP on different algorithms to find how the number of states affect things. The first size is 4x4, which results in 16 states. The second one size is 16x16, which results in 256 states. I think it's interesting because it is just like the grid-world example introduced in lectures. The number of states is different in two cases, one is smaller and the other is larger (256 states). Furthermore, the Transition Model make the action in each state very stochastic and unpredictable, which will result in the optimum policy very interesting.

1.2 Forest Management

In the Forest Management environment, there is a growing forest without grid. And the growth will reach a maximum. In each step:

MDP tuples ((S,A,P,R,γ)):

Action: CUT / WAIT ; CUT results in 1 reward and state turns to state-0 (can't cut, no more trees to cut); WAIT has two possible results: one with a probability p ($0<p<1$) passing to next-state; the other with *(1-p) probability* turns in state-0 (cause a forest fire).

So, what is interesting is that we have to decide if taking the risk that we could enter the final state to get the larger rewards or we just get 1 reward by cutting the trees at each state.

In the final state, if we select WAIT, we will get r2 reward back, also with *p probability*, will stay in the final state and with *(1-p)*, it turns in state-0 state; On the other hand, if we choose CUT, we will get r1 reward and directly goes back to state-0 (no trees to CUT).

2. Frozen Lake (16 states)

2.1. Value Iteration

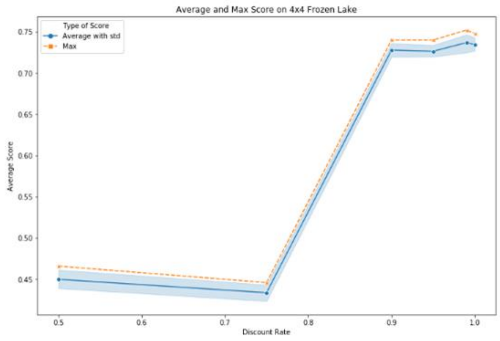
To answer the question on canvas, In Value Iteration, I have used epsilon value to define the convergence. So that, different discount-rates is very important because it will affect epsilon value and also the final values of non-terminal moves.

In the experiment, I have tried different discount-rates and different epsilon values.

I set Discount-rates: 0.5, 0.75, 0.9, 0.95, 0.99, 0.9999; epsilon values: 1e-3, 1e-6, 1e-9, 1e-12, 1e-15. These values are used in different tests. After algorithm converged, to test if they all converge to same answer, I have tested the resulting policies in 1000 episodes. Then, I get their mean reward.

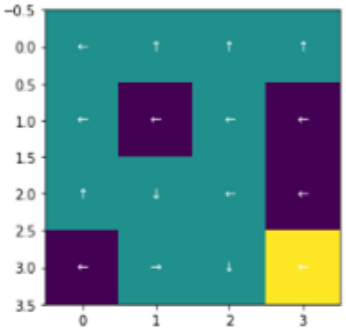
2.1.1 Effects of Discount-Rate

Discounts-Rate plays an important role in discounted reward. I realized that the higher discount-rates are, the better policies and also better rewards we will get.



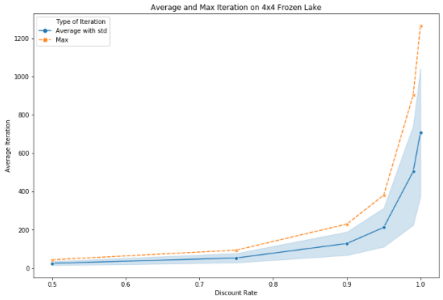
From the image, we can tell the orange one indicates the maximum value of the standard deviation, which is effected by different epsilon values.

It can be seen that a discount rate of more than 0.9 will lead to an average reward of more than 0.7. Furthermore, because some states are too far from the goal in the bottom right grid, their optimal values can't be assigned with the lower discount rates. Besides, when sigma is more than 0.9, the Value Iteration is good at assigning optimal values for states and creating good policy.

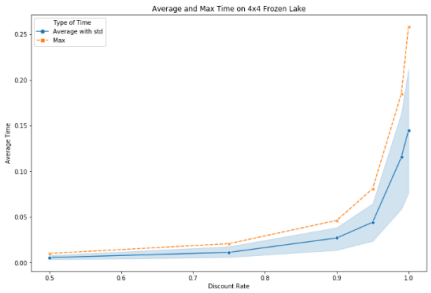


It's obvious that the policies in the problem instruct us to avoid all the holes and lead to the final goal at 75% of the time.

Take the time/iterations of convergence into account, I compare them (with different discount rate) as images below:



Iteration vs Discount Rate

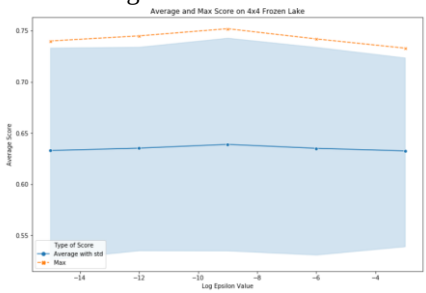


Time-Spent vs Discount Rate

It can be seen that the line of both iteration and discount rates or the Time-Spent and discount rates increase exponentially, which means high discount rates affects the time/iteration a lot. Considering the positive effect on rewards in this comparing tests, it seems reasonable that using a higher discount-rate will lead to better results. Moreover, after 1200 iterations of algorithm, the time spent in it is still around 0.25s. In short, I believe it makes sense to use a high discount rate in the 4×4 (16 states) environment.

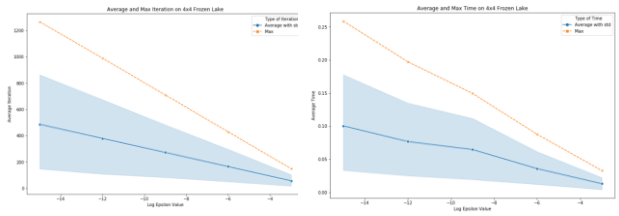
2.1.2 Effects of Epsilon Value

For the effects of epsilon value, we can see that the effects are hard to observe through the test, like below. Because the effects of discount-values mostly control the reward of testing in Value Iteration.



define epsilon using log-value at x-axis

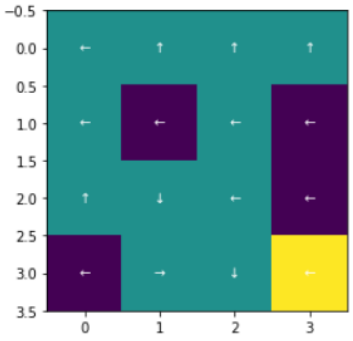
We can tell from the chart that different discount rates will lead to large standard deviation. And 1e-0 epsilon value can get the maximum reward, which is possibly caused by the randomization. Thus, if we use a epsilon value that is small enough, it's possible we will get a good policy.



Go back to convergence, we can observe the iterations/time cost from the images above. Different epsilon values result in different convergence iterations/time. When log value (epsilon value) is 1e-9, the policy gets the maximum reward with an average time near **0.5s**; on the contrary, when the epsilon value is 1e-15, it takes around 1s. For iterations, the condition is similar. Though the time spent in this case is small (0.5s, 0.1s), the phenomenon would make big difference when we solve other larger MDPs.

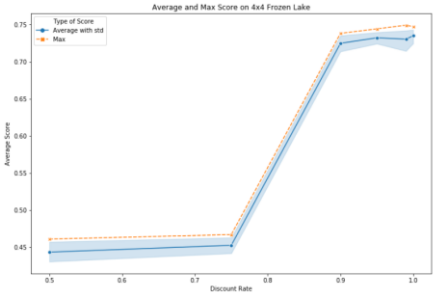
2.2. Policy Iteration

As mentioned above, Policy Iteration is used for analysis. In this part, I implemented discount rates and also epsilon and in the second step of Policy Iteration, different epsilon values are used and compared. I set Discount-rates: 0.5, 0.75, 0.9, 0.95, 0.99, 0.9999; epsilon values: 1e-3, 1e-6, 1e-9, 1e-12, 1e-15. After evaluation, the same policy comes out.

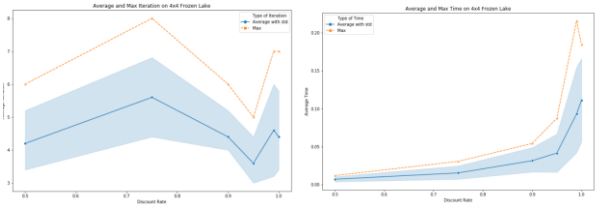


2.2.1 Effects of Discount-Rate

Because the resulting policy is just the same, the reward of Policy Iteration is similar to that of Value Iteration.



It can be seen that a discount rate of more than 0.9 will lead to an average reward of more than 0.7, whereas under 0.75 is about 0.45 only. Also, time spent and iterations are tested as below and we can compare them with those in VI.



For comparison, it can be seen that the number of iterations in Policy Iteration is much smaller than that in VI in the left-graph above. At the same time, the effects of discount rates are not that significant in VI. The maximum of iterations is 8, whereas VI took 1200 iterations to convergence.

However, for PI, we can tell that the discount-rate also influence the time in exponential way. But generally, the time spent for PI is smaller than that for VI, too. The max time of Value Iteration takes about 0.25s, while that of PI is 0.22s. In short, Policy Iteration takes a little less time (close) than Value Iteration.

2.2.2 Effects of Epsilon Value

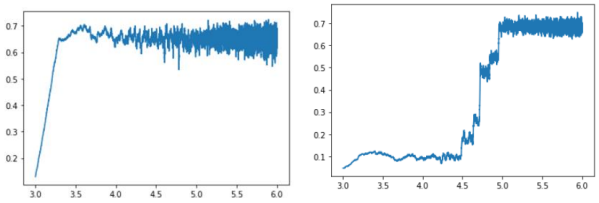
For policy, similarly, the epsilon value has less effect on it. But what is different is that epsilon value affects spent-time. I didn't put the epsilon's graphs on the paper because the epsilon values just display the randomness due to the discount rates. As I used a small enough epsilon value in this experiment, I believe that epsilon values only affect the spent-time. For example, lower epsilon value let the convergence of Policy Iteration longer, but the iteration number doesn't change by that as well as the average reward under the policy.

2.3. Q-Learning

For the exploration strategy of Q-Learning algorithm, I choose to explore some of the hyper-parameters and others not due to their weights: when I find some parameters are less effective, I assign them with constant values. I have used different values for training episodes, discount-rate, and learning-rate.

In detail, for epsilon-greedy, the epsilon value is set as minimum value in the linear decay. Only 1 discount rate (0.9999) is used at the 4×4 environment, tuning the variables. Meanwhile, for the convergence of algorithm, I compare the effectives of two constant learning rates with different values.

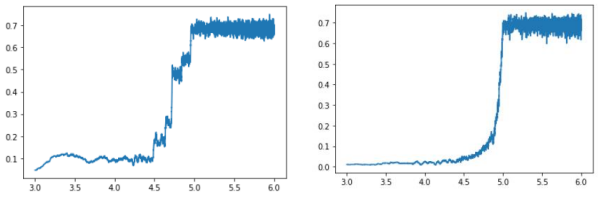
In order to observe the convergence, a rolling average value of reward is shown in the training phase. In order to compare the learning rates' effects, two graphs is shown below: the left one display the effect of 0.1 learning rate while the right one's learning rate is 0.01.



(x-axis is log normalized 6.0 means 1,000,000)

We can see the difference between the two graphs: First, with higher learning rate, it converged faster; on the other hand, the oscillation is more serious due to the accumulation of constant effects on the Q-table of each action; Second, a lower learning rate resulted in a slower convergence but it oscillates less. It's obvious that using a decay on the learning-rate (like we use on epsilon) can lead to faster convergence and less oscillation.

As the decay is used, I am going to find the effects of decay-rates on our epsilon. Next, I test epsilon values in linear decay with two decay rates. One is 1e-3, shown in the left graph below and the other 1e-5 one is shown in the right graph. Learning rate is 0.01.



(x-axis is log10 normalized)

As it shown above, the higher decay rate (1e-3) resulted in an earlier convergence (very slightly in the log10 normalization). In detail, both of the two reached the optimal value at around 100K.

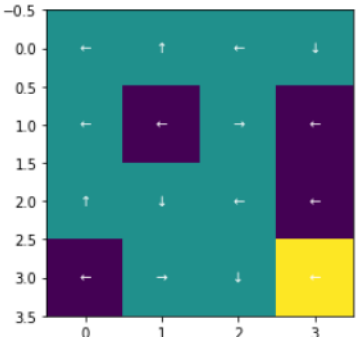
For the effects of decay rates, it seems like the value of decay-rate effects little in this highly stochastic environment. However, in other environments, in my opinion the decay rate will be as important as learning rate on exploration strategy.

In the table below, the different hyper parameters of Q-Learning and their effects on the reward as well as time spent are recorded:

	Discount Rate	Training Episodes	Learning Rate	Decay Rate	Reward	Time Spent
0	0.9999	10000.0	0.10	0.00100	0.747	8.363291
1	0.9999	10000.0	0.10	0.00001	0.726	2.830679
2	0.9999	10000.0	0.01	0.00100	0.108	2.436700
3	0.9999	10000.0	0.01	0.00001	0.290	2.924997
4	0.9999	100000.0	0.10	0.00100	0.591	104.188003
5	0.9999	100000.0	0.10	0.00001	0.546	53.979627
6	0.9999	100000.0	0.01	0.00100	0.205	46.713976
7	0.9999	100000.0	0.01	0.00001	0.747	48.768106
8	0.9999	1000000.0	0.10	0.00100	0.747	1120.504023
9	0.9999	1000000.0	0.10	0.00001	0.641	1013.372449
10	0.9999	1000000.0	0.01	0.00100	0.747	920.123316
11	0.9999	1000000.0	0.01	0.00001	0.731	988.899270

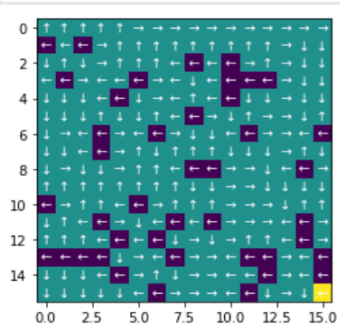
As it can be seen, with compared with PI/VI, Q-Learning takes a lot more time, in which the training episodes are the most effective factor. This is because we can't implement the model in the Q-Learning as we did in PI/VI and the exploration costs much time. For the average reward value, it seems like more training episodes lead to higher reward but there also are evidence for those policies with fewer episodes get over 0.74 mean reward. I think both higher learning rate and the stochasticity of Q-Learning algorithm help a lot on this. We can reach the optimal policy without an optimal setup.

In conclusion, we finally get the almost the same optimal policy no matter what algorithm we implement, Q-Learning, Policy Iteration or Value Iteration. The optimal policy is drawn below.



3. Frozen Lake (256 states)

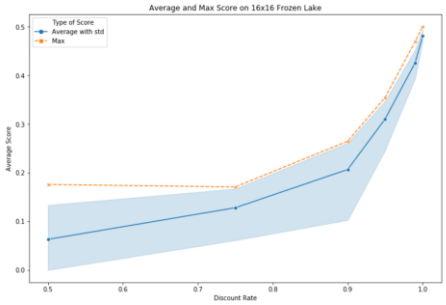
This section discuss the difference performance of the three algorithms mentioned above on the larger 16× 16 grid-world MDP problem.



First of all, as it can be seen, we are likely to get a lower mean of reward in the larger environment than that with smaller state size. This is because we have a longer way to move from start to the end.

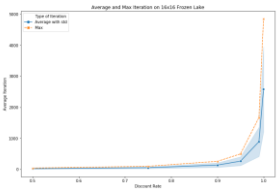
3.1. Value Iteration

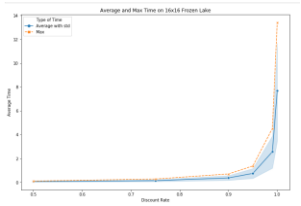
Like we did in the last section, the relationship between average reward and discount rates is discussed first.



From the graph, I found the maximum of the mean value of reward is now close to 0.5. And I noticed that now discount-rate's value affects the average reward more than it did in smaller environment. We can see that 0.999 makes significant difference comparing to 0.9. But in 4×4 grid-world, 0.9 discount rate is able to do a good job. So, the larger the environment is, the larger value of discount-rate we need for better average reward.

Then the time-spent and iterations part, we can come up with similar conclusion of discount-rate: higher discount rates effect more in a larger environment.

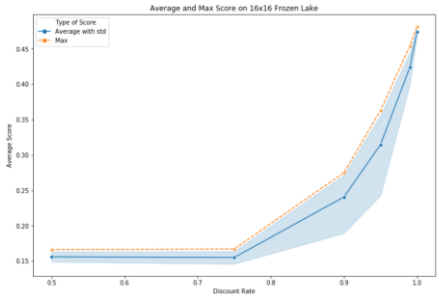




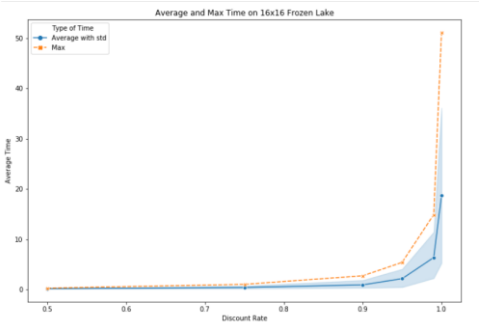
As it can be seen, a 256-state environment takes 14 seconds to converge, though it was only 0.25 seconds for a 16-state. This difference indicates that larger environments need much more time to converge. Similarly, same thing happens to the iterations, which increases from 1200 for a 4×4 to 5000 in this section.

3.2. Policy Iteration

In this section, PI for 16×16 environment came up with a same result of optimal policy as Value Iteration. Discount rate is 0.9999 and epsilon value is 1e-15.



From the Discount-Rate vs Average-Score graph, Policy Iteration performs better than Value Iteration when the discount rates are small. But, in general, it is exact that for PI, larger environments need higher discount-rate .



For the convergence time, tests for larger environment have reached a different consequence. PI tends to spend much more time to converge on a larger environment. The number of iterations is relatively small. The reason is that each iteration includes much evaluation of a number of states in a larger environment.

In conclusion, in 256-state environment, both VI and PI came up with similar rewards and same policy. And Policy Iteration spends more time than Value Iteration does to converge.

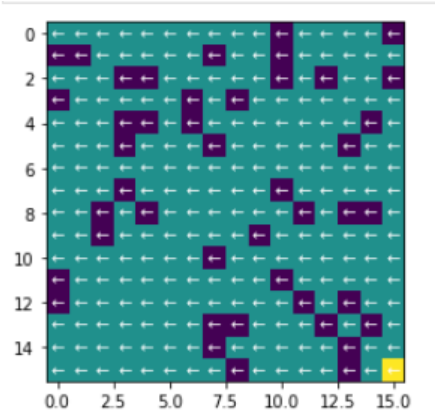
3.3. Q-Learning

It seems challengeable to implement Q-Learning on the large environment because there are too many holes, which make the risk of falling in them much more higher.

The table displaying multiple hyper-parameters plays the same role as it did in 4×4 environment.

	Discount Rate	Training Episodes	Learning Rate	Decay Rate	Reward	Time Spent
0	0.9999	10000.0	0.10	0.00100	0.0	0.710954
1	0.9999	10000.0	0.10	0.00001	0.0	1.609952
2	0.9999	10000.0	0.01	0.00100	0.0	0.851520
3	0.9999	10000.0	0.01	0.00001	0.0	1.773063
4	0.9999	100000.0	0.10	0.00100	0.0	6.428837
5	0.9999	100000.0	0.10	0.00001	0.0	12.025417
6	0.9999	100000.0	0.01	0.00100	0.0	7.981473
7	0.9999	100000.0	0.01	0.00001	0.0	11.076562
8	0.9999	1000000.0	0.10	0.00100	0.0	62.562974
9	0.9999	1000000.0	0.10	0.00001	0.0	67.106997
10	0.9999	1000000.0	0.01	0.00100	0.0	66.184676
11	0.9999	1000000.0	0.01	0.00001	0.0	73.556381

To my surprise, the results have shown that all the average reward of policies are 0. What's more, the time spent values are much lower than they are in 16-state. To figure out what happened, I check the Q-table, which indicates that all the training tests of exploration lead to an ending that we enter a hole. No one get the 1 reward of managing to arrive the final goal. So, that's why all the values are 0.



As we can see it is obvious that getting to the final goal in the stochastic world is not easy. If no one success to reach the goal, the value in the q-table will stay 0. Especially the actions are random, making it more complex. In that case, PI/VI, which have a actions model, are much better in larger environments with only 1 reward at the end.

I consider that if we give some little rewards to staying alive or set -1 reward for entering the holes, maybe the algorithm would attempt to avoid the holes. Or, it might helps if we set a random start grid instead of the top left one? I would like to try.

4. Forest Management (20 states)

In this MDP, the discount-rate has is no longer a hyper-parameter but conducted as a decision of design. We can't only depend on the discount rate to decide cut or wait because each action results in different reward, instead of only 1 reward at the goal like the last MDP. Since the discount rate will lead to distinct next-environment, it's better to set a constant value for it. In this section, discount rate is 0.9.

As the situation that if we always start as state-0, we cannot move on and the rest of test can't continue, which is meaningless. In order to make testing better, I start as each state for 1000 times, coming up with different average reward at the end of the policy. Meanwhile, since the discount-rate won't change, it also can be used for the rewards. Thus, in the testing environment, when the rewards keep same all the time, the second reward will be 90% of the first one and the third will be 90% of the first one, and so on.

I set r1 = 6 and r2 = 10 for the Forest Management.

4.1. Value Iteration

To answer the question on canvas, In Value Iteration, I have used epsilon value to define the convergence. So

that, different epsilon values: 0.1, 0.01, 1e-6, 1e-9, 1e-12, and 1e-15 are used.

Table below shows the relationships between parameters and rewards.

	Epsilon	Policy	Iteration	Time	Reward
0	1.000000e-01	(0, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, ...	31	0.002649	2.154275
1	1.000000e-03	(0, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, ...	52	0.002633	2.156900
2	1.000000e-06	(0, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, ...	85	0.005243	2.142954
3	1.000000e-09	(0, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, ...	118	0.006456	2.152037
4	1.000000e-12	(0, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, ...	151	0.007815	2.185030
5	1.000000e-15	(0, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, ...	186	0.010476	2.113817

In general, we get the same policies and all the rewards are close to each other, even we start from a random state. Another thing, when epsilon gets smaller, it takes more Iterations and longer times, as we expected. Furthermore, Value Iteration runs very fast no matter what the epsilon values are in all cases.

4.2. Policy Iteration

In policy iteration, since I have used mdptoolbox's [3] implementation in this problem, I couldn't test for different epsilon values and got only 1 run using the constant discount-rate.

Policy Iteration gave the same policy (so similar mean-reward) with any of the Value Iterations and did this in 12 iterations and 0.03 seconds. As it can be seen, Policy Iteration took the longest time comparing any of the VI above.

For smaller Frozen Lake environment, PI took close (a bit shorter) time comparing to VI. And it solved Frozen Lake in 8 iterations (4x4) at most. But in this environment, it took 12 iterations. That is because in this sort of chain set-up, PI changes a smaller part of the policy each iteration instead of correcting many at the Grid World set-up.

4.3. Q-Learning

Since Q-Learning uses an agent to explore the environment, this type of chained-state environments seems difficult for it. So, I have explored 5 different hyper-parameters for it to converge to the optimum state and checked all their mean-rewards and time to complete the training.

I've explored "number of iterations", "epsilon", "epsilon-decay", "learning-rate", and "learning-rate-decay" hyper-parameters. The results are as in the table below.

As it can be seen, all the resulting policies get close results to each other but when checked, I realized that the policies are quite different than each other even though much of them are similar. Also, none of the 32

:0]:

	Iterations	Alpha Decay	Alpha Min	Epsilon	Epsilon Decay	Reward	Time
0	1000000	0.990	0.0010	10.0	0.990	2.112963	43.618149
1	1000000	0.990	0.0001	10.0	0.990	2.114554	43.677410
2	1000000	0.999	0.0010	10.0	0.990	2.123520	43.546746
3	1000000	0.999	0.0001	10.0	0.990	2.088811	43.096997
4	1000000	0.990	0.0010	10.0	0.999	2.140274	43.282926
5	1000000	0.990	0.0001	10.0	0.999	2.149241	43.344919
6	1000000	0.999	0.0010	10.0	0.999	2.111441	43.098411
7	1000000	0.999	0.0001	10.0	0.999	2.125839	43.276635
8	1000000	0.990	0.0010	1.0	0.990	2.149132	43.359320
9	1000000	0.990	0.0001	1.0	0.990	2.152381	44.149231
10	1000000	0.999	0.0010	1.0	0.990	2.135872	44.541396
11	1000000	0.999	0.0001	1.0	0.990	2.170419	44.624184
12	1000000	0.990	0.0010	1.0	0.999	2.188515	42.555094
13	1000000	0.990	0.0001	1.0	0.999	2.134553	43.361591
14	1000000	0.999	0.0010	1.0	0.999	2.136003	45.035051
15	1000000	0.999	0.0001	1.0	0.999	2.152113	44.596225
16	10000000	0.990	0.0010	10.0	0.990	2.089561	437.497844
17	10000000	0.990	0.0001	10.0	0.990	2.132088	464.760277
18	10000000	0.999	0.0010	10.0	0.990	2.133040	470.177954
19	10000000	0.999	0.0001	10.0	0.990	2.093046	477.440789
20	10000000	0.990	0.0010	10.0	0.999	2.154250	482.488369
21	10000000	0.990	0.0001	10.0	0.999	2.112108	472.839738
22	10000000	0.999	0.0010	10.0	0.999	2.114887	512.676572
23	10000000	0.999	0.0001	10.0	0.999	2.155651	578.868911
24	10000000	0.990	0.0010	1.0	0.990	2.163981	579.066845
25	10000000	0.990	0.0001	1.0	0.990	2.105347	554.939058
26	10000000	0.999	0.0010	1.0	0.990	2.129672	533.451718
27	10000000	0.999	0.0001	1.0	0.990	2.134965	495.792148
28	10000000	0.990	0.0010	1.0	0.999	2.095265	498.717053
29	10000000	0.990	0.0001	1.0	0.999	2.121986	490.025702
30	10000000	0.999	0.0010	1.0	0.999	2.156720	507.424240
31	10000000	0.999	0.0001	1.0	0.999	2.151777	585.571571

policies are equal to VI/PI policies (which are same).

On the other hand, the time spent for the Q-learning is much longer. It can take up to 10 minutes, whereas VI/PI takes 0.03 seconds at most.

Iterations	Alpha Decay	Alpha Min	Epsilon	Epsilon Decay	Reward	Time
1000000	0.9945	0.00055	5.5	0.9945	2.136602	43.697768
10000000	0.9945	0.00055	5.5	0.9945	2.127772	508.858674

After checking the mean values for 2 different iteration value, I saw that the reward didn't increase from 1M to 10M iterations, which means the algorithm already converged in 1M, but time has increased by 10x, as expected.

I have checked the mean values of other hyper-parameters, as well. But the results are so similar to each other on both reward and time. This means that no matter what hyper-parameter I have used, it has converged to a good enough policy.

5. Forest Management (500 states)

To analyze differences between a small and larger non-grid problems, I have used 500 state Forest Management environment. In this environment, I kept using 0.99 discount-rate but since it is a very long path to reach to final state to collect the rewards, I increased r1=100 (cut at final state) and r2=15 (keep at final state).

5.1. Value Iteration

In Value Iteration, I have used the same epsilon values with 20-state Forest Management problem (0.1, 0.01, 1e-6, 1e-9, 1e-12, and 1e-15).

	Epsilon	Policy	Iteration	Time	Reward
0	1.000000e-01	(0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, ...	79	0.069786	2.744159
1	1.000000e-03	(0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, ...	119	0.037132	2.738842
2	1.000000e-06	(0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, ...	179	0.199475	2.736443
3	1.000000e-09	(0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, ...	239	0.058018	2.726877
4	1.000000e-12	(0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, ...	299	0.387262	2.749473
5	1.000000e-15	(0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, ...	348	0.363754	2.724796

It can be seen that the mean-reward that policies extract from the environment are pretty close to each other in this environment, too. This means that 0.1 epsilon value is small enough to get the best policy. But, the number of iterations is doubled comparing to 20-states, which is pretty reasonable and the time spent is multiplied with 36 for 500-states.

I think these numbers are pretty good from 20 to 500, we can say the time is almost linear with the number of states. That means VI works pretty well in larger environments, too. This was seen in the Frozen Lake environment, as well. VI didn't increase its time so drastically, comparing to PI, with regard to the state size.

5.2. Policy Iteration

The Policy Iteration algorithm created a policy with 2.72 mean-reward, which is so close to what VI creates. But the chain property increased PI's iteration and time-spent in 500-state environment, as well. It took 46 iterations and 3.1 seconds to converge for PI, which is 10 times longer than what VI took with its most extreme epsilon value. The underlying reason is again the fact that policies are dependent to each other. When next state's policy changes, then it matters more for the current state change or not. This increases the iteration and the time.

As it was seen in Frozen Lake environment, the time and iteration of PI increases much more with regard to the size of environment, comparing to VI. Also, the chained-state environment makes things more difficult for PI. But, in the end it is able to get the optimum policy in 3 seconds in a 500-state environment.

5.3. Q-Learning

Forest-Management with 500 states seems to be a difficult problem for Q-Leaning to solve. But the biggest advantage here is that the agent gets some reward in almost every state. So that, it will have more guidance.

I have used the same 5 hyper-parameters and got this table from Q-Learning on Forest Management.

	Iterations	Alpha Decay	Alpha Min	Epsilon	Epsilon Decay	Reward
0	1000000	0.990	0.0010	10.0	0.990	0.802000
1	1000000	0.990	0.0001	10.0	0.990	2.577495
2	1000000	0.999	0.0010	10.0	0.990	2.631859
3	1000000	0.999	0.0001	10.0	0.990	2.646176
4	1000000	0.990	0.0010	10.0	0.999	2.539574
5	1000000	0.990	0.0001	10.0	0.999	2.606902
6	1000000	0.999	0.0010	10.0	0.999	2.569366
7	1000000	0.999	0.0001	10.0	0.999	2.637456
8	1000000	0.990	0.0010	1.0	0.990	2.670379
9	1000000	0.990	0.0001	1.0	0.990	2.690222
10	1000000	0.999	0.0010	1.0	0.990	2.619189
11	1000000	0.999	0.0001	1.0	0.990	2.651154
12	1000000	0.990	0.0010	1.0	0.999	0.820000
13	1000000	0.990	0.0001	1.0	0.999	2.679801
14	1000000	0.999	0.0010	1.0	0.999	2.650142
15	1000000	0.999	0.0001	1.0	0.999	2.707013
16	10000000	0.990	0.0010	10.0	0.990	2.723851
17	10000000	0.990	0.0001	10.0	0.990	1.026000
18	10000000	0.999	0.0010	10.0	0.990	2.756234
19	10000000	0.999	0.0001	10.0	0.990	2.818315
20	10000000	0.990	0.0010	10.0	0.999	2.707936
21	10000000	0.990	0.0001	10.0	0.999	2.835022
22	10000000	0.999	0.0010	10.0	0.999	2.739897
23	10000000	0.999	0.0001	10.0	0.999	2.788664
24	10000000	0.990	0.0010	1.0	0.990	2.802476
25	10000000	0.990	0.0001	1.0	0.990	2.865915
26	10000000	0.999	0.0010	1.0	0.990	2.651490
27	10000000	0.999	0.0001	1.0	0.990	2.761586
28	10000000	0.990	0.0010	1.0	0.999	2.762742
29	10000000	0.990	0.0001	1.0	0.999	2.851094
30	10000000	0.999	0.0010	1.0	0.999	2.764619
31	10000000	0.999	0.0001	1.0	0.999	2.812812

In this environment, it can be seen that different hyper-parameters affected the policy and the resulting reward significantly. Some of the policies could get less than 1.0 mean reward, whereas some of them surpassed

the VI/PI algorithms. When analyzing independently, grouping by different hyper-parameters:

Iterations	Alpha Decay	Alpha Min	Epsilon	Epsilon Decay	Reward	Time
1000000	0.9945	0.00055	5.5	0.9945	2.406171	57.976636
10000000	0.9945	0.00055	5.5	0.9945	2.666791	513.044849

It can be seen that 10M iterations gets significantly better result, unlike 20-state Forest Management. So, a larger state space needs more training. This makes sense because Q-learning learns by interacting the environment. From this table, we can say that Q-learning needs more than 1M iterations to converge in this environment.

But, the most significant effect is caused by "learning rate decay" hyper-parameter:

Alpha Decay	Alpha Min	Epsilon	Epsilon Decay	Reward	Time
0.990	0.00055	5.5	0.9945	2.372588	286.648156
0.999	0.00055	5.5	0.9945	2.700373	284.373330

A higher decay, which causes a slower decrease in the learning rate results in a much better policy. This can be explained by the same fact. It takes longer time to explore an environment with more states. If learning-rate drops too much before the agent reaches to certain states enough times, the q-table can't be updated with a large enough learning rate.

Comparing the q-learning to VI/PI in this environment, we see that it never comes up with the same policy with them. But it gets good enough policies that get over 2.8 reward, whereas PI/VI can get around 2.74.

6. Conclusion

Two different environment and three different algorithms are used in this analysis. Two of the algorithms were model-based (PI and VI) and they had a big advantage to produce the best policies since they interact with the model directly. This is supported by the results on both mean-reward their policies got, and the time-spent for training.

Q-Learning, a model-free algorithm, has to interact with the environment instead of interacting with the model. So that, it is more difficult to explore the environment for this algorithm. We have witnessed this in Frozen-Lake 16x16, where Q-Learning didn't even reach the Goal state in 10M training episodes. So that, it got no value in the q-table. This can be improved by different techniques as explained in that section but its disadvantage is obvious.

When comparing the grid vs non-grid world environments, we can see that grid world is easier to converge for Policy Iteration because of the Forest Management environment's chained-state nature.

On the other hand, we have seen that increasing the state size doesn't affect the model-based algorithms significantly. They can still solve the problem in reasonable iterations and they reach the optimal. But this case is not true for Q-learning.

7. References

[1] Gym: A toolkit for developing and comparing reinforcement learning algorithms. (2020). Retrieved 11 April 2020, from https://gym.openai.com/envs/FrozenLake-v0/

[2] Sutton, R., & Barto, A. (2018). Reinforcement learning (2nd ed.). London, England.

[3] Sawcordwell. (2015, April 13). sawcordwell/ pymdptoolbox. Retrieved from https://github.com/ sawcordwell/pymdptoolbox/