Markov Decision Processes

Trevor Goodell
OMSCS
Georgia Institute of Technology
Atlanta, USA
tgoodell6@gatech.edu

***Abstract*—This paper explores the field of Markov Decision Process (MDPs) by looking at two problems and how they behave as the number of states grow.Each problem will be solved with two classic algorithms, Policy Iteration and Value Iteration. As well as a reinforcement learning algorithm Q-Learning.**

***Index Terms*—MDP, Frozen Lake, Forest Management, Policy Iteration, Value Iteration, Q-Learning**

I. MDPs

A. Frozen Lake

The Frozen Lake problem consists of a boy playing with a ball, and his friend accidentally kicks it onto a frozen lake. The boy is tasked with retrieving the ball. However, the ice is very slippery and there are thin spots of ice randomly across the lake. It is our job to find the steps he should take to get his ball safely. The frozen lake is a "Grid World" problem, meaning the boy travels to each grid using the 4 cardinal directions. A visual of the environment can be seen in figure 1.

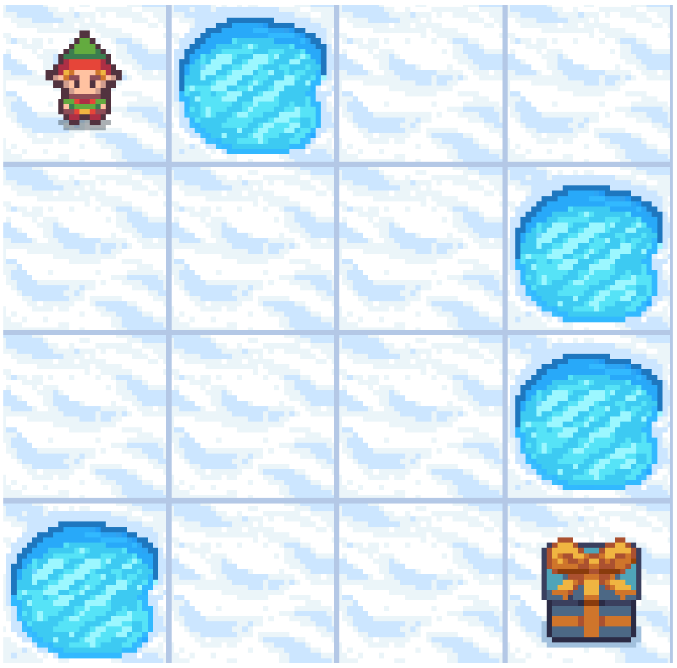


Fig. 1: Frozen Lake Environment

By looking at this problem, you might think that this is quite easy as all you would need to do is not step in the hole. And that is where you would be wrong! With each step, you have a probability of $\frac{1}{3}$ to move in your intended direction, however, you have a chance to move in either of the two directions perpendicular of your intended action, both with probability $\frac{1}{3}$. For example, if in our starting state shown in figure 1, if we took the "Down" action, we would have a $\frac{1}{3}$ chance of moving to the empty square south of us, a $\frac{1}{3}$ chance of not moving because running into a border results in you staying put, and a $\frac{1}{3}$ chance of us falling into that whole on our right.

The states are pretty simply labeled, the boy is standing in state 0. The top row is labeled [0, 1, 2, 3] and each square is labeled +4 from the square above it. Therefore, the present is in state $3 + 4 + 4 + 4 = 15$. The reward for being in any state is 0, unless you step into the state that contains the present, in which case the reward is 1. The Frozen Lake environment is brought to you by OpenAI Gym, Gym is a toolkit for developing and comparing reinforcement learning algorithms [1], more on them later.

B. Forest Management

For the sake of Professer Isbell's and Littman's variety itch, we have switched away from a Grid World problem. This problem is called the Forest Management problem, and we are responsible for ensuring the ethical harvest of wood from a forest. The forest grows trees until it reaches full capacity, each time step is corresponding to, let's say, waiting a day for our extremely genetically engineered trees to grow in a matter of days. The forest reaches capacity after S time steps, where S is an input to the problem.

At each time step, you have the two actions of "WAIT" or "CUT". If you "CUT", you will harvest the forest and return to state 0, where the forest is empty. If you "WAIT", you will advance to the next time step with probability p. With probability $1-p$, there will be a forest fire in which case you return to state 0 without harvesting any wood.

When you perform the "CUT" action, you will receive 1 as a reward unless you are in state S-1, in which case you will receive $r2$. If you perform the "WAIT" action, you will receive a reward of 0, unless you are in state S-1, in which case you will receive $r1$. Both $r1$ and $r2$ are inputs to defining the MDP.

II. Solving Frozen Lake

The Frozen Lakes problem is relatively simple to setup but it will offer good insight to strengths and weaknesses are. Here, since the professors imply that size matters for MDPs, the Frozen Lake environment will go from 16 states to 4096 states. This will clue us into the number of iterations, wall time, and consistency of each of our algorithms as the number of states grow.

A. 4x4

The 4x4 Frozen Lake environment was no issue for any of our algorithms. I have shown the 3 policies generated by each of our algorithms in figure 2. The Blue square is our starting point, the yellow square is the state that contains the toy, the green states are the holes, and the purple states are the thick ice.

Here we can see that Policy Iteration and Value Iteration converge to essentially the same policy. In Value Iteration, you initialize your policy to 0, which in this environment equates to the action LEFT. This representation was largely just an error in adding the logic to the visualization algorithm. An ”X” in any non green square indicates that every action is equivalent.

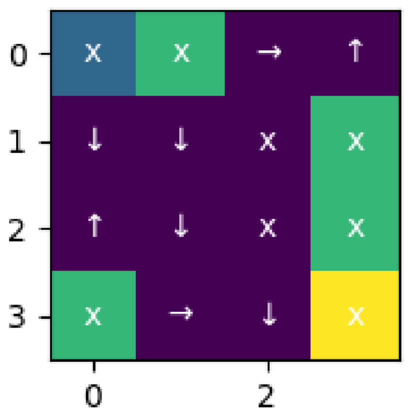
Some interesting things to note is the behavior that is learned when around holes. All 3 algorithms learn that it is better to perform ”safer” actions. These actions are the ones that result in you not ending up in a hole. In state 3, the top right corner, you can see it wants you to go ”UP”, this has a $\frac{2}{3}$ chance of you ending up in the same state. More on this in the conclusion section.

Q learning converges to a slightly different solution than both PI and VI. In c1r2 (column 1 and row 2), VI and PI choose the ”DOWN” action while Q-Learning choose the ”RIGHT” action. With my all knowing power of this environment, I believe that the ”RIGHT” action is a slightly worse solution for this tile. The main difference is the states that you end up in if you don’t go in your intended direction. By going ”DOWN” in c1r2, you either end up in c0r2, c1r3, or c2r2. These 3 states have higher values than the 3 states you would end up in by choosing the ”RIGHT” action. Specifically ending up in c1r1 is significantly worse than ending up in c0r2, which are the only differences between the two actions.

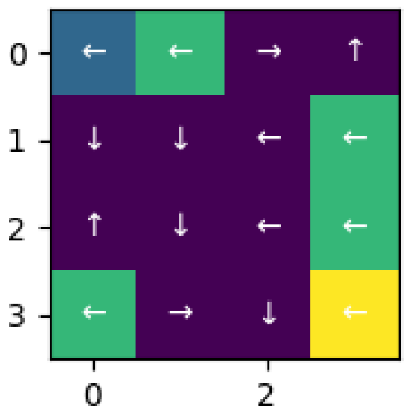
In terms of time, the results are quite interesting, shown in Table I. You can see that Policy Iteration and Value Iteration converge in nearly the same time while Value Iteration takes 90 times as many iterations. This is due to Policy Iteration makes much more complicated updates than Value Iteration at each step.

	Iterations	Time (s)
PI	3	0.029
VI	271	0.028
QL	9902161	171.963

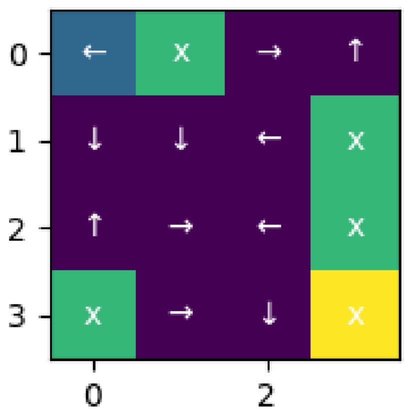
TABLE I: 4x4 Time Information



(a) Policy Iteration



(b) Value Iteration



(c) Q-Learning

Fig. 2: 4x4 Policies

Policy Iteration has 2 components to it; policy evaluation and policy improvement. In policy evaluation, it evaluates the value of the policy at each state at updates it through a Bellman operator. Then in policy improvement, it tests to see if the update changes its best action. It converges once the changes stop. Value Iteration combines those two steps into one by looking over all possible actions in the update state. Then, it runs until the changes are below an arbitrarily small threshold.

Q-Learning is a bit deceiving here. Q-Learning is drastically different than Policy Iteration and Value Iteration. Those 2 algorithms look at every state and action combination and figure what is optimal. Q-Learning starts by blindly walking

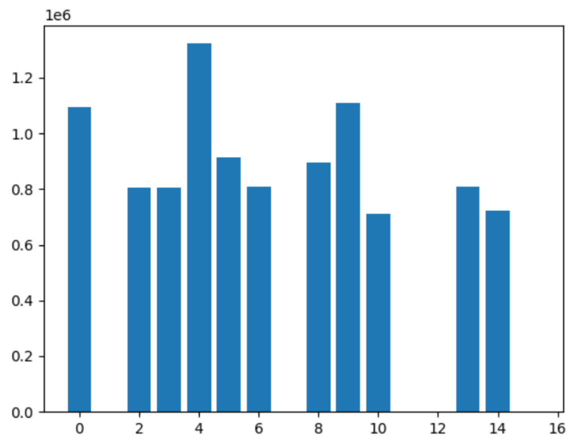


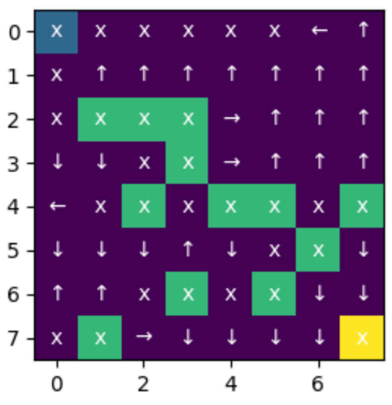
Fig. 3: Visited States

around with random steps and slowly learning the environment. I considered a step in the environment as an iteration, since the Q table (which would be the equivalent of the Value table from PI and VI) is updated on every step. It also trains for at least 250000 episodes, therefore the average number of steps in an episode is about 40. This is slightly concerning, considering that it takes 6 steps to reach the end state and there are 13 states that are non-terminating. This means that on average, the learner is stepping on every tile 3 times per episode. In fact, you can see the histogram of what states were visited by the learner during training in figure 3. Very interesting observations include state 4 being the most visited, since it's the only way out from the starting zone. And state 14 being the least 14 being one of the least visited considering it's the only way into the reward state.

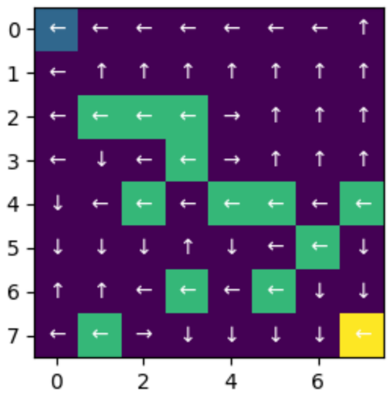
B. 8x8

The 8x8 results are actually much more interesting than the 4x4 ones. Figure 4 shows the policies obtained by each algorithm.

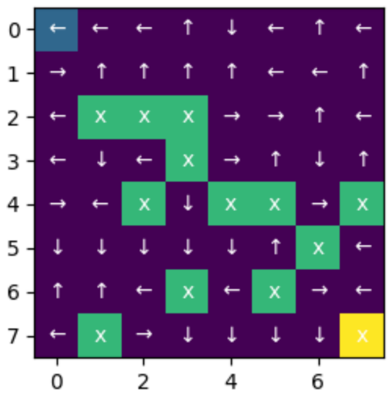
Again, PI and VI are very similar, but there is one difference that is bizarre. In columns 0 and 1 and rows 2 through 4, there is a slight difference between VI and PI.c0r2 and c1r4 are what I am going to call "trap states". In these states, all though they are not episode ending states, you don't have any good option. Therefore, it is interesting to see how each of the algorithms try to maneuver around it. Since PI can look one step ahead, it knows what will happen if it goes into c0r2 or c1r4 and it makes moves to avoid them. Therefore, it chooses "DOWN" for c0r3 and "LEFT" for c0r4. VI does not have the ability to look one step ahead. All it can do is look at the value of its neighboring states. Figures 5 and 6 show the values of each state for PI and VI, respectively, and darker colors indicate lower values. You can see that the value of c1r3 is lower for VI than PI with respect to it's surroundings, which is why VI chooses to ignore it and PI doesn't mind entering it.



(a) Policy Iteration



(b) Value Iteration



(c) Q-Learning

Fig. 4: 8x8 Policies

Q-learning is significantly different than the other 2 algorithms. Overall, it's much worse than the policies formed from PI and VI. One notable example is c6r3, the left and right states equal if you choose "UP" or "DOWN", however c6r4 is drastically worse than c6r2 so going "DOWN" is a much worse

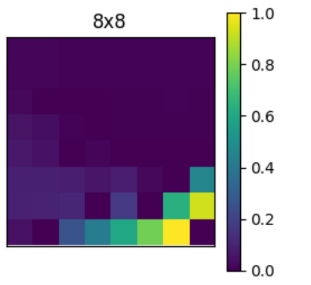


Fig. 5: PI Values

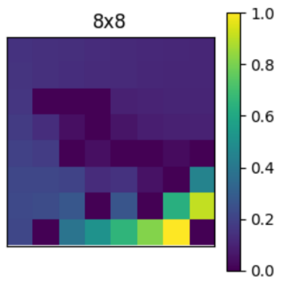


Fig. 6: VI Values

	PI	VI
4x4	0.028	0.029
8x8	0.488	0.165
16x16	1.494	0.834
32x32	9.283	4.26
64x64	59.614	16.097

TABLE III: Frozen Lake Times

	PI	VI
8x8	17.4	5.7
16x16	3.1	5.0
32x32	6.2	5.1
64x64	6.4	3.8

TABLE IV: Frozen Lake Time Increases

than going ”UP”. The main path to the solution (going down c0 and then navigating to c2r7 and going right) is relatively similar with Q-learning and the other algorithms.

The time for convergence is shown in table II. The results are quite different than what we saw in the 4x4 case.

	Iterations	Time (s)
PI	4	0.488
VI	355	0.165
QL	14285347	279.447

TABLE II: 8x8 Time Information

There, VI and PI were nearly identical in time. Here they are different by a factor of 3, showing that the greedy approach is much better for larger problems. The number of iterations increased as well, however they all increased by roughly 30 to 40 percent. The time for Q-Learning to converge nearly doubled. Number of training episodes and max number of steps remained the same. However, the state space is much larger, so much more room to walk around. Average number of steps increased by 44.2 percent, while time to converge increased by 62.5 percent. I would have expected these numbers to be much more in line since each step takes the same amount of time.

C. 16x16, 32x32, and 64x64

To save the grader a bit of time from staring at more of these policy maps, they will be located in the repo associated with this assignment. In general, PI and VI converged to similar solutions with more and more slight variations between each other. The main purpose of including these larger state sizes was to explore the time complexity of the algorithms and then verify it. For this section, I will be ignoring Q-Learning. It will be discussed again in the conclusion section, but for now all you need to know is that Q-Learning fails to converge on these larger maps. And the time results are irrelevant to this section.

Table III shows the times, in seconds, each algorithm took to converge on their respective maps.

Table IV shows how much the time increased as the map increased. This means that for policy iteration, the 8x8 map converged 17.4 times slower than the 4x4 map

There are a few issues with this analysis that I want to address before continuing. The maps aren’t consistent, meaning that the 4x4 map could be very easily solved while the 8x8 map is much more difficult. This leads to some inconsistencies. To better improve these results, I would need to run these experiments quite a few times and then average the results.

Anyways, let’s first take a look at the two algorithms provided in Sutton and Barto’s book ”Reinforcement Learning”. First value iteration, it has one main loop where it iterates over every state and checks what states it can go to based on what actions. This leads to a computational complexity of $O(|S|^2|A|)$. For policy iteration, there are two loops, the first being policy evaluation where it performs a similar algorithm to that of value iteration, leading to a complexity of $O(|S|^2|A|)$ for policy evaluation. However, for policy improvement it checks if the policy has any changes. This loops over the policy which has size $|S|$ and then checks it against the values in the transition table which has size $|S|^2$, therefore the complexity of policy improvement is $O(|S|^3)$. Since both loops are quadratic loops, neither dominates the complexity therefore the complexity of policy iteration is $O(|S|^3 + |S|^2|A|)$.

These results explain why policy iteration takes more time than value iteration at all time steps, the added $O(|S|^3)$. These results indicate an upper bound on our time, which makes sense as all of our values are strictly less than what they should be.

D. Conclusion

One reason why Q-Learning averages so many steps inside the MDP is that there is no cost for walking around. Each state has a reward of 0 and since Q-Learning’s objective is to maximize it’s reward, adding a ton of 0’s would have no affect. This concept can help self-driving cars drive more efficiently. The ’reward’ for accelerating is a faster speed, but it has a cost of burning gas. Therefore, it can be punished for accelerating excessively. By adding a reward at each step that is slightly negative, the agent would be pushed to finding the solution faster.

Another reason why Q-Learning struggles in the larger maps is that it cannot find the reward in the amount of steps given. It would always fall in a hole before reaching the end. Since stepping in the holes have a reward of 0 the agent never learns to avoid the holes. In the 16x16 environment, by the time epsilon decayed to a point where it was highly likely that it would follow it's own policy, it had never reached the end and every state was worth 0. If you look at the policy maps for the Q-Learner at the larger map sizes, you will see that every state has an "X" for it's grid. One thing that would alleviate this would be to have a large negative value for stepping into a hole. This would teach the agent to first avoid the holes, and this would lead the agent to finding the reward at the end.

In general, the policies formed by Q-Learning were slightly worse than PI/VI. This is to be expected. PI/VI are able to look at every possible state, action, and next state pair and determine the optimal solution from there. Q-Learning must discover every state, action, and next state pair, and then to converge to the true value of that state, must visit it multiple times and perform multiple different actions from it. This is not as bad as it may seem however. PI and VI struggle to perform in partially observable MDPs where not every state is known, yet the Q-Learning agent will maintain a model of its environment and add to it as the states are discovered.

III. Solving Forest Management

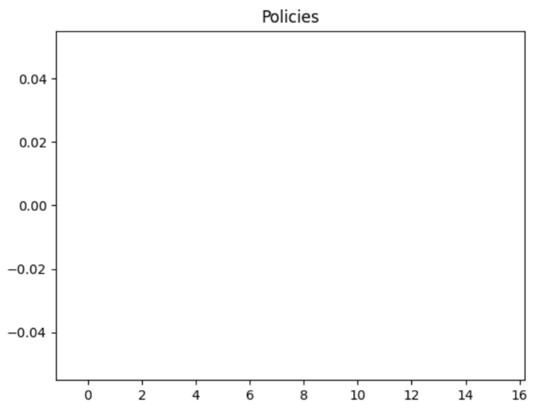
For these experiments, I set p to 0.9, $r1$ to 4, and $r2$ to 1.

A. Policies

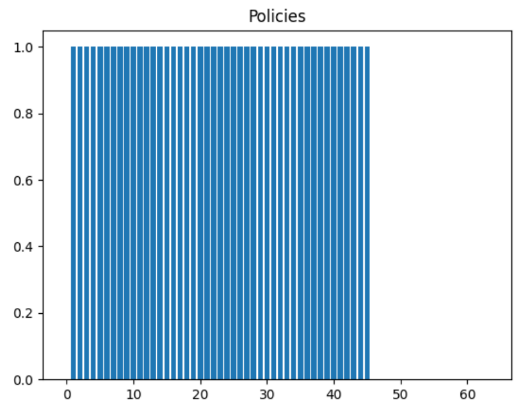
For the policy section, states 1024 and 4096 will be ignored as they don't provide a lot of information. Here PI and VI produce the exact same results so only one graph will be included.

Here you can see in figure 7a absolutely nothing! This means that the optimal action in every state is 0 which is the "WAIT" action. When you increase the number of states to 64, you can see the optimal policy is 1, aka "CUT", until S = 45. This was interesting to me as it can be viewed as "CUT" unless you are in the last 18 states. If you apply this to figure a, it follows the same rule. Except there are only 16 states so that's why they're all 0. To verify these results, I checked against the next size up of 256 and the results are the same! It performs "CUT" for every state until S=237. If you do check on states 1024 and 4096, the same result holds. This verifies that the policy of "Cut unless last 18 states then wait" holds. This leads me to believe that this is essentially an expected value problem.

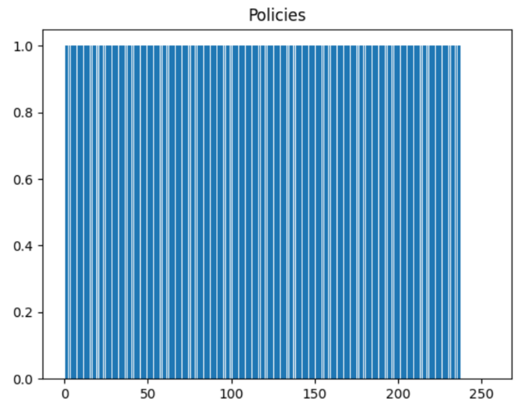
Unfortunately, Q-Learning comes in and throws a loop into this whole paradigm. Looking at figure 8 you can see this is a bit sporadic, but you can see that unlike PI and VI, it chooses "CUT" much more often than "WAIT". This is due to Q-Learning having to choose how much it values future rewards. With improper tuning of parameters, Q-Learning won't be patient and will aim for immediate rewards.



(a) 16 States



(b) 64 States

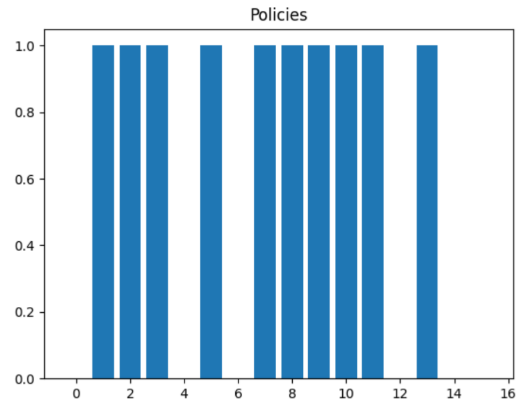


(c) 256 States

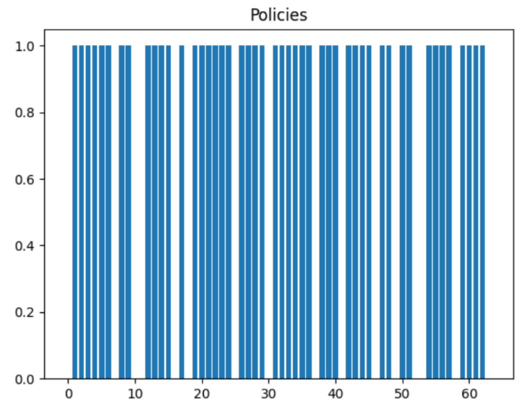
Fig. 7: Policies

B. Values

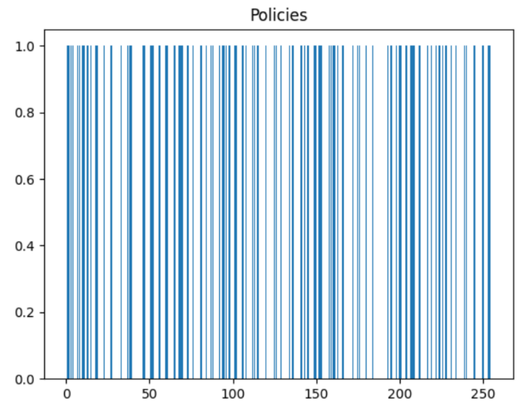
The values have very similar shapes as number of states increases, for discussion purposes we will only be using the values from 64 states. Some notable findings is that the value of state 0 is slightly lower than the flat part for PI and VI. This is because you have nothing to cut in state 0, therefore you have to wait, hence the policy being 0 for every state 0 in the previous section. Therefore, you have to wait to get to state 1, but then you have a chance of a forest fire occurring. The flat part corresponds to where the policy was all equal



(a) 16 States



(b) 64 States

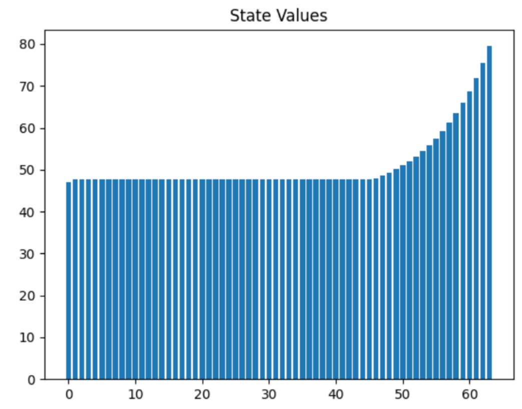


(c) 256 States

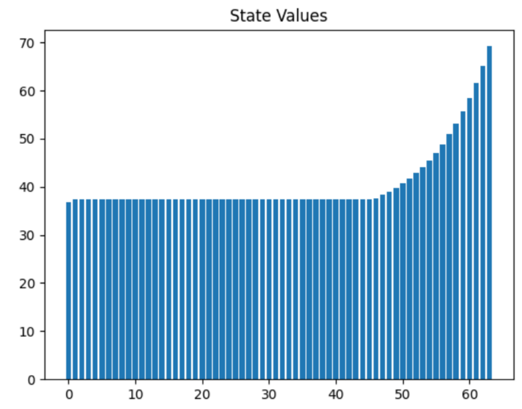
Fig. 8: QL Policies

to 1. This is because the reward for cutting is the same at all states. Therefore, any state that you cut in is identical to the state before it. Finally, the exponentially increasing ramp corresponds to the "WAIT" part of the policy. This makes sense, the highest value state is the final state and the discount rate applies exponentially as you get further away.

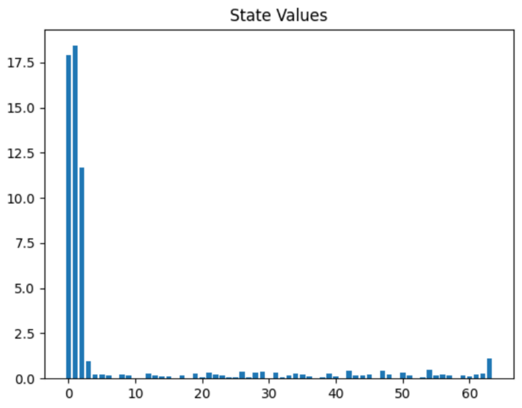
For Q-Learning, the values are just as sporadic as the policy. But, the lower states are much higher value than the higher states. The value of the 0 state is still lower than the following states. This also shows that the Q-Learner does not value the later rewards that PI and VI can check on.



(a) Policy Iteration



(b) Value Iteration



(c) Q-Learning

Fig. 9: Values

C. Times

The times are a lot less interesting than in the Frozen Lake problem. This is mostly due to this being a much simpler MDP to solve, but the results are shown in table V. The increases as number of states increases is a lot more sporadic than in the Frozen Lake env, but consistently between 2 and 10.

Num States	PI	VI	QL
16	0.002	0.0005	1.435
64	0.12	0.003	1.581
256	0.234	0.02	2.116
1024	1.305	0.042	4.285
4096	13.118	1.437	13.342

TABLE V: Convergence Time (s)

D. Further Experiments

After running the tests to generate the above data, I was relatively underwhelmed with the results. I then played with a few of the parameters to see how it would affect Policy Iteration and Value Iteration.

1) Effects of R1: R1 is the reward you get for waiting until the end. I wanted to see what would happen if this value changed from 4, while keeping r2 the same. Using r2 = 2, the same pattern of ”CUT unless you are in state S-N” holds until r1=0.5. When r1 reached a low enough value, then it would always CUT unless you were in state S-2. This is because you can either cut for 1 and go to state 0, or wait and with probability $p = 0.9$ get to the final state and then cut and receive r2. Otherwise, increasing R1 shifts the start of WAIT earlier and earlier. These results were consistent across PI and VI.

2) Effects of R2: R2 is the reward you get for cutting at the last state. R2 had no effect on the policy. This is because when you are in a state where you want to wait, you will keep waiting until a forest fire occurs. R2 is only in affect when you are in the last state, and choose to cut, which would never happen since there is no chance of ”accidentally” cutting when you intended to wait.

3) Effects of P: Unsurprisingly, changing P only changes when you would want to start WAIT. A lower p moves it earlier, and larger p moves it later. This makes sense, if you are not worried about a forest fire you will keep waiting. If they are frequent, you wouldn’t risk it.

4) Effects of gamma: Both PI and VI use gamma as a hyperparameter and this controls how much to discount future rewards. Again, the more you value future rewards, the more you will be willing to wait and risk. By increasing gamma, it moves the WAIT policy earlier and earlier.

E. Optimizing Q-Learning

I knew Q-Learning could work if it had some tuning. Therefore, I tried to setup a makeshift GridSearch and test some different hyperparameters. Unfortunately, the grid search didn’t yield much better results, even over drastically different parameters. The results are going to be shown in the FM3 directory in the repo associated with this assignment.

IV. Overall Conclusion

Overall, PI and VI performed significantly better than Q-Learning. This was to be expected since the number of states were still computationally reasonable. The natural exploration done by Q-Learning which could eliminate useless states was never needed in these problems. As previously mentioned, these MDPs were fully observable which is another condition where PI and VI succeed. In terms of wall clock time, VI and PI were fast, with VI always being faster than PI. Q-Learning took significantly more time than PI. In terms of iterations, PI always took the least, followed by VI, and finally Q-Learning.

The largest improvement I would make to these problems is including negative rewards. I think a negative reward would greatly help the Q-Learner perform better, especially on the Frozen Lake problem. Negative rewards are also realistic in the real world, there is usually a cost associated with interacting in an environment.

References

[1] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016.