Assignment 4:
Markov Decision Processes

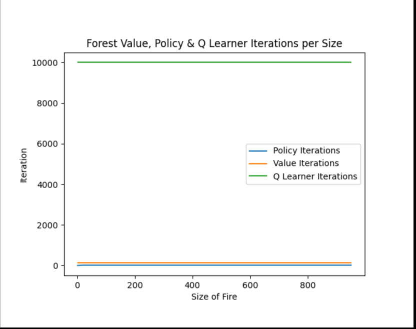
Oindril Dutta | odutta3@gatech.edu

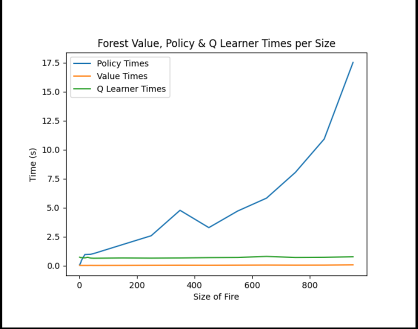
1 INTRODUCTION

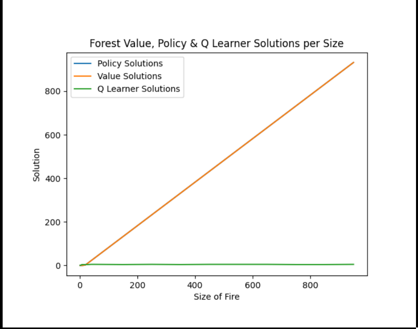
This assignment was about Markov Decision Processes (MDP) and how the third way of learning - reinforcement learning (RL) - could work in the real world. It's an agent-based learning technique - giving the agent information and letting it decide what's the best course of action. This paper is about exploring a few different ways to do that. Specifically, I use Value Iteration (VI), Policy Iteration (PI), and I picked Q Learning (QL) as my favorite reinforcement learning algorithm of choice. For the MDP problems, I picked Forest Management (FM) and Frozen Lake (FL). I decided to use FM as the small vs large problem space comparison due to its simplistic problem statement.

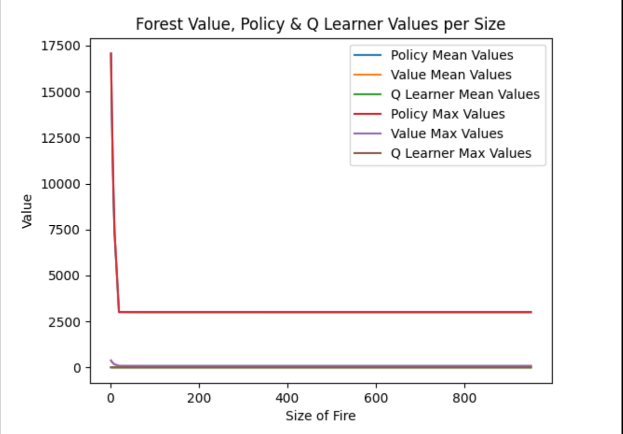
2 Forest Management

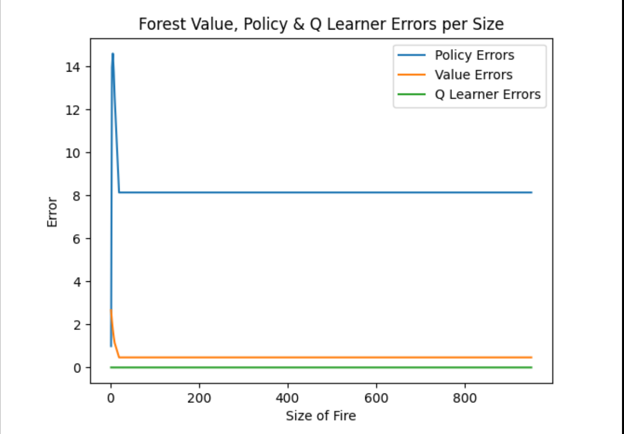
The forest management problem (FM) is pretty interesting because it's about trying to get an agent to be able to effectively gamble. There are two different options for the agent when managing a forest - either cut it down or wait and let the forest grow. There's a higher reward for waiting and letting the forest grow and a smaller reward for cutting it. That's because it's in the best interest of the world to let the forest grow. However, if there's a fire, it would negatively harm the forest, and it can only happen when the forest isn't cut - so the agent has to balance keeping the forest alive long enough collecting rewards versus cutting it down and not letting it get consumed by the fire. Interestingly, the finite horizon nature of this problem effectively determines the agent's behavior. When making charts, I decided to make charts over multiple sizes of the problem, from 2 all the way to 1000 to see how different MDPs solve the problem differently at different sizes. To do this I set the seed, as usual, and then generated forests of varying sizes, and then collected how many iterations, how much time, how much error and rewards the MDPs used at different sizes. I also specifically collected something I'm calling Solution Count that allows me to gauge how the MDP behaved when solving the problem. Specifically, the solution count is the number of times the MDP decided to cut the forest. I decided to plot all of the different MDPs - PI, VI, and QL on the same charts to make it easier to compare all of them on things like iterations, time, errors, rewards, and solutions. I could do this because I'm comparing sizes, not iterations, which I do compare in Frozen Lake. This methodology gives me five charts below to analyze:











2.1 Value Iteration vs Policy Iteration

Policy Iteration versus Value Iteration is pretty interesting. It seems like regardless of the problem size, both can perform really well. However, PI does seem to have some trade-offs. It has slightly fewer iterations at smaller problem sizes, and while iterations stay the same, the time it takes to converge is much higher at larger problem sizes compared to smaller problem sizes. It's faster than VI and QL at smaller sizes, but it grows slower much more quickly. In terms of actual PI convergence, it's pretty similar to VI. However, it does seem to be able to get a higher value reward when it comes to the max reward PI can get, especially when compared to VI, much higher than VI and QL. However, it also has much more error compared to QL and VI, especially at smaller sizes. But both VI and PI seem to have pretty stable behavior at larger problem sizes, at least for this forest management problem. It takes only a couple hundred iterations for VI to converge, whereas it takes policy even less to converge. They seem to have the same rate of convergence, but policy definitely does it slower, at least when it comes to real-world time. This is interesting because I was assuming it would be the exact opposite - PI has a smaller space to work through compared to VI because VI has to work through an arbitrary and sometimes continuous space to find convergence. However, PI has a finite space to find convergence in, because the number of possible policies is finite. So it's interesting, but, actually, it makes perfect sense! The larger the problem size, the larger the policy space PI has to navigate through to find a good solution! So it ends up taking more time than VI for a larger problem space; maybe for a more complicated problem, maybe a continuous MDP problem, PI might be better because VI would have a much larger space to sift through to converge. That is very interesting, and makes a lot of sense! That would also explain the higher error at smaller sizes for PI because it has a smaller set of possibilities it can look through to converge, and the best possible solution with lower error might not be in that smaller limited policy space.

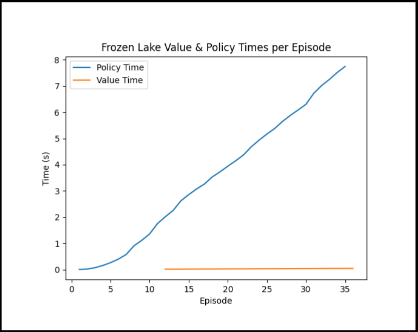
2.2 Q Learner

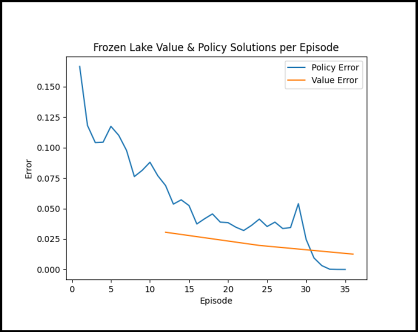
My QL implementation comes from the MDP toolbox. It's an epsilon greedy Q Learner, tuned with a low epsilon, high gamma, and a high alpha with default decays. I observed this configuration to have the best performance in a grid search. Looking at the charts, I can make a few different points of analysis for QL - PI and VI don't take that many iterations to converge, QL however, seems like did not converge even after 10,000 steps regardless of the problem size - but I don't know if that's true. It's a little confusing, because when it comes to error, the Q Learner has almost no error, whereas VI and PI do have errors, which may mean QL has converged, and it's just the best QL can do. Or maybe error has nothing to do with convergence. Regardless of that, it doesn't seem like the QL is able to learn this problem well because it doesn't have the ideal solutions and rewards when compared to VI and PI. It consistently has very few rewards. I know what the best solution is because I set the chance of fire to be very low - 10% - which is the MDP problem default and so it makes more sense for the learner to cut less, which PI and VI do as you can see in the Solutions chart. However, at the very beginning of the graphs, the QL does do better than PI and VI when it comes to solutions. But that doesn't hold true for the other charts and increasing the number of episodes also didn't help QL converge faster or get better results, it just ended up giving the same results for more iterations and time. For me, this indicates that QL might need exponentially more iterations and steps to learn, converge and have better performance at larger problem sizes, which is a little concerning because PI and VI do much better when it comes to finding convergence and high rewards. It might also be a problem with MDPToolbox's specific QL implementation.

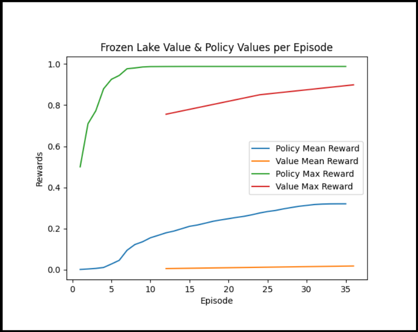
3 Frozen Lake

The Frozen Lake MDP problem is a pretty simple grid world problem. It becomes interesting however when you consider its slipperiness, metaphorically and literally speaking. The learner agent is somebody trying to traverse an ice sheet-covered lake, which is very slippery, to get to the goal, apparently retrieving a Frisbee. However, on the way, there are multiple holes in the lake ice sheet which could lead them to fall in the cold, cold winter water, which no learner would want. The slipperiness comes into play when the agent is deciding to move - when moving in any specific direction, they could literally go in a perpendicular direction instead, for example going North might instead slip the Agent East or West, in either direction, 2/3 of the time! Thus only a third of the time the agent can actually go in the direction they decided to go. It's a very stochastic grid world MDP problem. It's very similar to the problem that was brought up in the lecture, however a more exacerbated version. So I think it's very interesting because the agent has to actually learn that when it wants to go in a specific direction, perhaps towards the goal, it shouldn't go in the obvious direction, it will have to skirt that slipperiness. At first glance, it's going to move in weird directions, but when considering the probabilities of everything would make sense. A great example of this is two holes 1 tile apart from each other - it's a bit of a worst-case scenario! Because to cross that gap going straight through it, you would need to be lucky with 1/3 odds. It actually makes more sense for the learner to head directly towards the hole! Then there would be a 2/3 chance of NOT falling in the hole, escaping the situation by either passing the gap or going backward without falling in the hole to try again, with only a 1/3 chance to fall in the gap.

The methodology and implementation for this problem were much simpler compared to the Forest Management MDP problem, even though it's technically a more complicated problem with a larger problem space. All I did was set up the MDP from the open AI gym library, generate a random map of a specific size, and then let it run with different solvers, VI, PI, and QL. I was able to create six charts. Three for the time, reward, and error vs episodes for Policy and Value Iteration, and the other three with the same axes, but only for Q Learner. Please excuse the typo in the graphs saying "solution" instead of "error" in the title, the Y label is correct. I simply plotted the error, time, and reward, against the episodes that the MDP learners ran for. I couldn't plot the QL lines with the PI and VI lines in the same charts due to the massive discrepancy in the number of iteration episodes for the QL. The lines for the charts come directly from the MDP toolbox runtime statistics after running the problem through the VI, PI, and QL solvers that were tuned with a GridSearch.



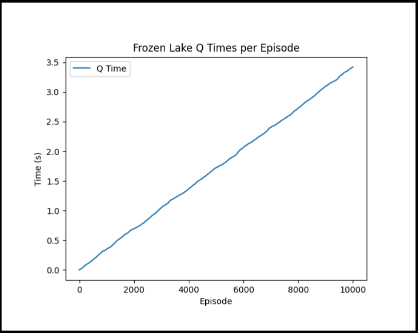


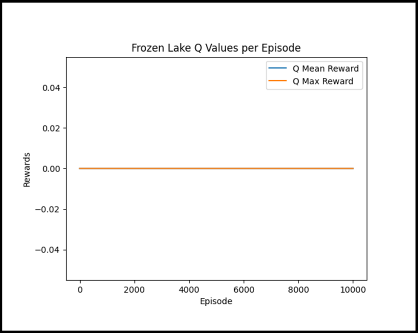


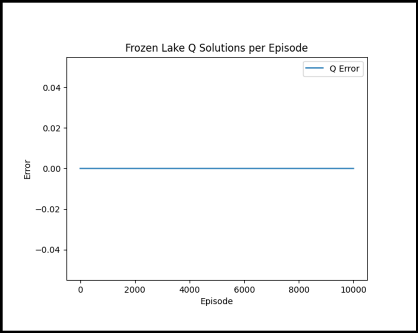
3.1 Value Iteration vs Policy Iteration

Following the analysis from the Forest Management MDP problem, the same analysis and insights can be applied to the differences between VI and PI. VI takes less time and has roughly the same solutions, but takes more iterations and doesn't achieve the same level of reward value as PI can.

3.2 Q Learner







The QL did not perform well on this problem at all, and I think I can point to a few specific reasons. When it comes to a breakdown of rewards, this problem is actually not that well set up. There's only a reward when the agent reaches the goal and there seems to be no negative reward anywhere to help guide the agent. I have to manually add a function that checks to see if the agent session ended when it reaches a goal or a hole part of the performance issues with this problem, which I will touch on further later on. I believe is due to this absorbing state in the form of holes that have no negative rewards, so the learner agent doesn't actually learn to avoid them. In fact it burns many iterations trying to get to the goal only getting any reward that it can propagate back in the history once it reaches the goal. So potentially speaking, the agent could not have any idea of how to behave until it happens to luck upon the goal because there are no rewards other than the goal itself for smaller maps. This isn't a problem because with enough iterations it will eventually reach the goal.

However, with larger problem sizes, apparently 15x15 and above, it does not do well at all because the learner never actually reaches the goal and continuously burns iterations. Just trying to get its first semblance of learnings from a reward in the future. I definitely want to change the reward structure to better reflect learnings for the learner agent. I would love to do something like a -1 reward for falling in a hole and maybe some sort of distance-based penalty - the closer the agent is to the goal the smaller the penalty would be, something a hundreth the size of the reward, also like the lecture, to give some urgency to the agent. A distance penalty might backfire though, because there may or may not be a path from a close-by tile on the map to the goal because it might be blocked by holes, that the agent might not consider. Again, it seems like I would have to run it with a lot more iterations and fix these problems in future work to get the Q Learner to perform well. VI and PI did not have these problems because it wasn't searching from the perspective of the agent.