Markov Decision Processes

Stephen Shepherd | sshepherd35@gatech.edu | 04/17/2022

Introduction

In this paper I will explore three approaches to solving Markov Decision Processes (Value Iteration, Policy Iteration and Q-Learning) as applied to a small stochastic grid world maze problem and a large stochastic non-grid world problem of Forest Forest Management.

Implementation

I used the "hiive" MDP Toolbox's implementation of the algorithms. I modified the Q-Learner algorithm to not "teleport" to a new state after 100 iterations. For the Maze problem, I stopped iterations if the goal state was reached.

The Problems

1) Maze / "Semi-Frozen Lake"

I designed a small 15 x 15 (225 total) state Maze environment with stochastic Manhattan movement rules for the agent: .9 probability the desired direction is taken and .05 probability for the direction's two adjacent right angles. The maze contains a ring of pitfall -30 reward states with a small opening far from the reward and a large opening near the reward, and an intermediate wall of discouraging but survivable -7 reward states blocking a direct path to the reward from a large portion of the states. The goal state is in position [7,14] and contains a reward of 20 and is modeled as terminal in the transitions and rewards setup. Other states contain a small negative reward of -1 to encourage the agent to find its way to the goal quickly. I did not designate a starting position as I wanted to focus on learning a policy for starting in any state. Discount rate was .99 unless otherwise noted. There are two components in this environment that pose interesting challenges for the algorithms:

1. Avoiding the ring of pitfall -30 reward states, potentially traversing the risky shortcut in [7,2]
2. The wall of -7 reward states, which could be worth traversing under certain conditions

This problem is interesting as it poses some risk/reward tradeoffs to consider and presents an opportunity to easily visualize how the algorithms approach the problem in the 2x2 state space.

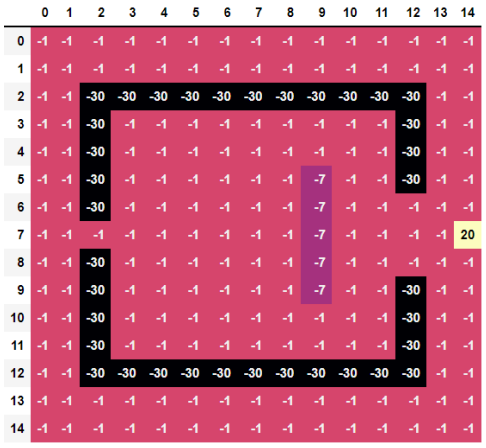


***Figure 1**: The Maze visualized with rewards for each state.*

2) Forest Management Scenario

I utilized MDPToolbox's Forest Management setup for 1,000 states. The scenario presents the possibility of a large reward in the forest's oldest (last) state, compared with small rewards for harvesting the forest for firewood (then proceeding back to state 0) in any state except the first. Each state represents a timestep (eg. a year), and there are two actions: cut for firewood (1) or don't cut (0). Every year, there is a one in one thousand (.001) probability of a fire destroying the entire forest, in which case the agent transitions to state 0 (the youngest state). Discount rate was .995 unless otherwise noted. The rewards are set up as follows:

- **Wait**: 0 for every state except the last state in which it is 4. This represents a large government payout for having let an older forest stand as a wildlife refuge for a while.
- **Cut**: 0 in state 0, 2 in the last state, and 1 in every other state. This represents a small payout for selling the forest cuttings for firewood.

It's important to note that in the last state, not being terminal, either action sends the agent back to state 0. In this manner, the Wait reward in the last state can only be achieved by not cutting in the preceding states (allowing the agent to enter the final state), and future states and actions (after the reset to state 0) are incorporated regardless of which action is taken in the last state. This problem is interesting as the fire component introduces a risk consideration that needs to be balanced with the large conservation reward opportunity in the last state. It is a nice contrast to the Maze problem because there is no terminal state and it is over 4x larger with 1,000 states.

Table of Results

Problem	Num States	Algorithm	Run Seconds	Mean Value	Mean Cumulative Reward
Maze	225	Value Iteration	0.1	4.29	4.19
		Policy Iteration	0.2	4.29	4.19
		Q-Learning	32	4.29	4.12
Forest Management	1,000	Value Iteration	0.5	169	1035
		Policy Iteration	9.3	169	1044
		Q-Learning	700	87	501

Maze - Value Iteration and Policy Iteration

I ran Value Iteration (VI) and Policy Iteration (PI) with a Discount factor of .99 to determine a solution Policy for the maze problem. Max Iterations for Value Iteration was determined using the span norm analysis described by Puterman[1] combined with an Epsilon of .01. The initial Policy for Policy Iteration was set as that which maximizes the expected immediate reward, computed via the Bellman Equation. Both approaches produced equivalent policies, and produced similar Mean Cumulative Rewards (4.19 for both VI and PI) and Mean Value (4.29 for both VI and PI). Mean Cumulative Rewards were simulated using 10k episodes each starting in a random state with iterations for each episode terminating at the earlier of 1k or reaching the goal state. PI only took 5 iterations, while VI took 48. However, VI only took .01 seconds to converge while PI took .02 seconds.

[1]M. L. Puterman, *Markov Decision Processes*, Wiley-Interscience Publication, 1994

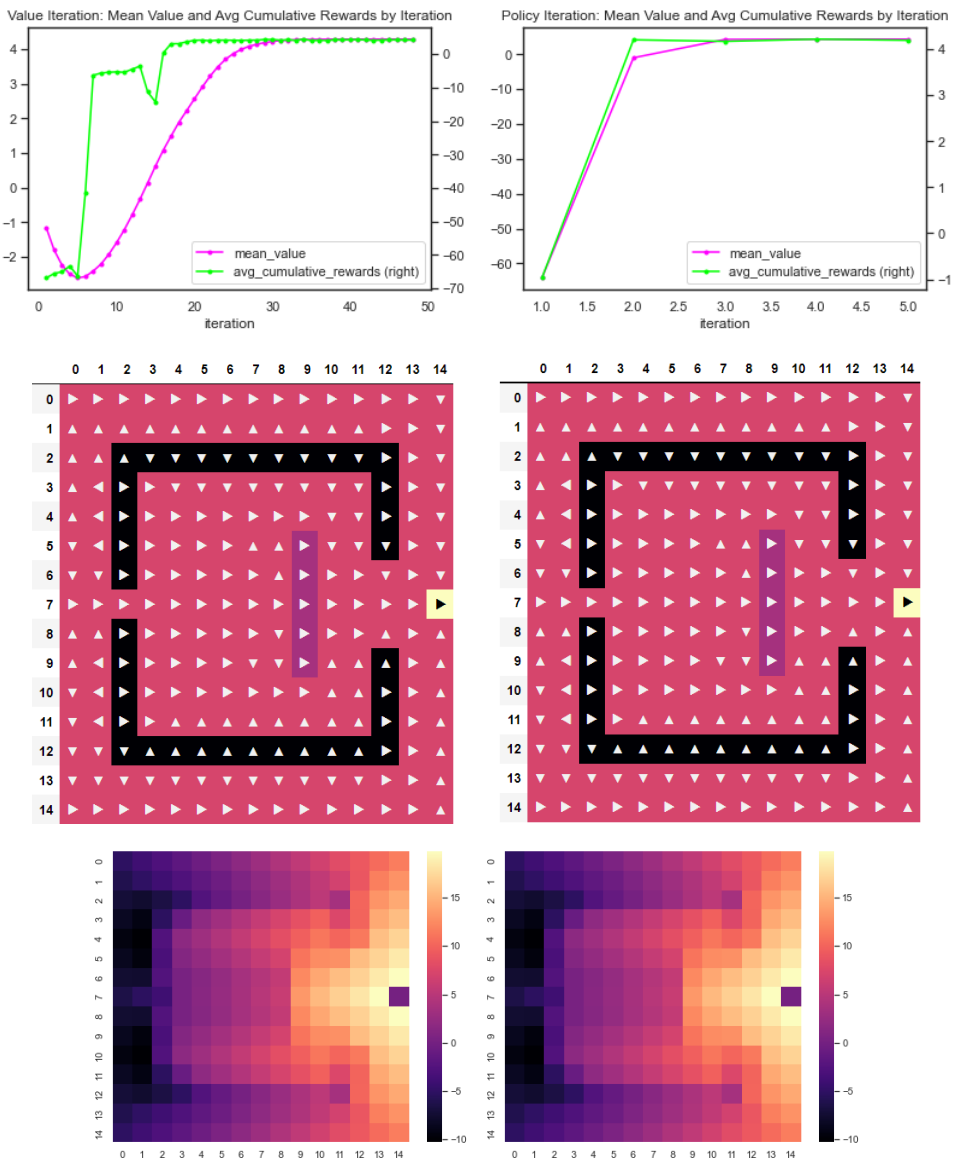


***Figure 2**: Value Iteration (left) and Policy Iteration (right) results visualized. The middle image represents the policy, while the bottom image represents state value.*

The policies appear intuitive: they mostly avoid the pitfall -30 states and drive toward the goal. Interestingly, at the discount factor of 0.99 the policies elect to take the risky routes through positions [7,2] and [7,9]. Position [7,2] is risky to traverse because the stochastic right angle movements offer the possibility that the agent would fall into the -30 rewards directly above and below when attempting to travel east. Position [7,9] offers the agent the chance to avoid traversing seven -1 reward states to go around the wall at a cost of -7. The policies prefer these riskier routes at a discount of 0.99 because the reward, when reached, will typically still be high enough to justify the risk. Using a discount factor of 0.8, however, both algorithms produce policies that avoid these routes, as compared in *Figure 3*. The .8 discount policy prefers to avoid more immediate (and less discounted) negative rewards instead of taking an expensive shortcut to a reward that will be diminished in a few timesteps. The policy at 0.99 discount also "hugs corners" at [2,1] and [12,1] instead of giving them a wide margin. The Mean Cumulative Rewards for the 0.8 discount runs (-1.61 for VI and -1.55 for PI) reflect the discount and are similar. The

policy for position [7,7] varies between Value Iteration and Policy Iteration but doesn't appear to impact the overall outcome and is probably due to differences in action selection in a tie-breaking scenario.

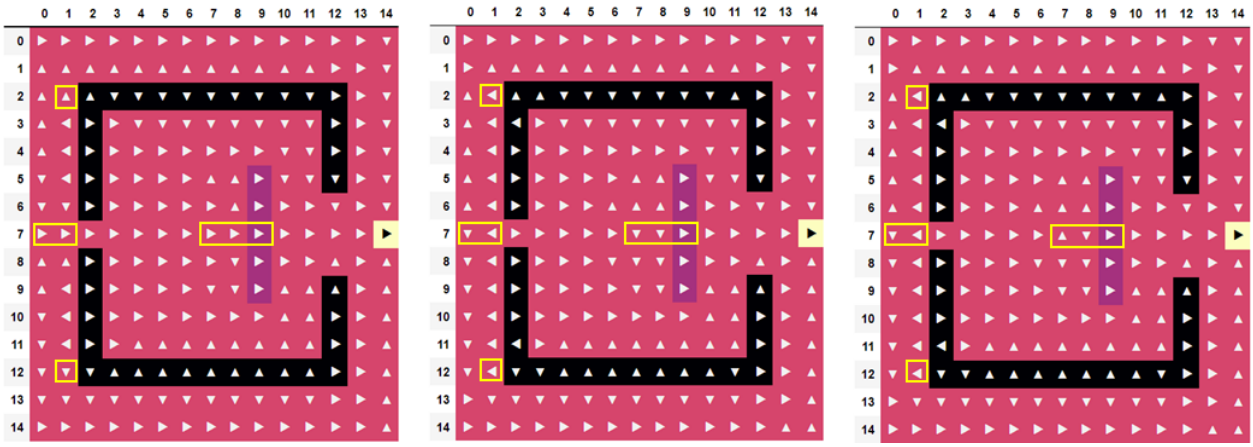
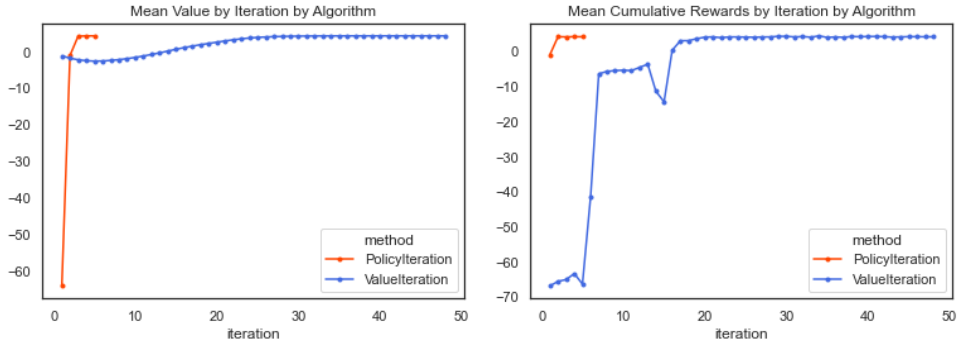


***Figure 3**: Policies by Discount Factor. Left: VI and PI policy using 0.99 discount. Middle: VI policy using 0.8 discount. Right: PI policy using 0.8 discount.*

Value and Policy Iteration show opposite behavior at the start of the iteration curves for Mean Value and Mean Cumulative Rewards: Policy Iteration starts with extremely low Mean Value and improves a lot, while Value Iteration starts with extremely low Mean Cumulative Rewards and improves a lot. Examining the value heatmaps in *Figure 4*, we see that Value Iteration's first iteration produces highly local values whereas Policy Iteration's first iteration cascades the value from the goal state farther into the map. So, VI's iteration 0 doesn't offer a path to the goal for most states, whereas PI's does right off the bat. This explains why VI's Mean Cumulative Reward is so low in the first few iterations. That being said, PI's iteration 0 offers little guidance away from pitfall states in the western, northern and southern edges of the map, which explains why its Mean Value is so low in the first iteration. By the last iteration for each algorithm, the state values look extremely similar. VI appears to do a better job at identifying the negative reward states to avoid locally early on, although it takes more iterations for it to learn to give them a wide margin in case of stochastic movement error.



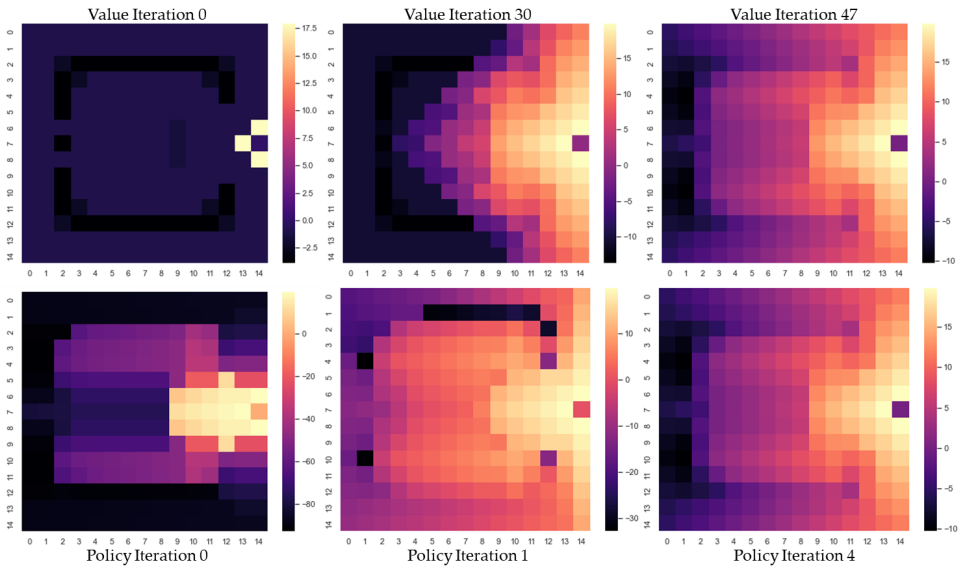


***Figure 4**: Value Iteration vs. Policy Iteration by Iteration. Top plots show Mean Value and Mean Cumulative Rewards by iteration. Heatmaps show state Value by algorithm for selected iterations.*

Maze - Q-Learning

For Reinforcement Learning, I ran Q-Learning at 10k episodes with per-episode stopping if the iteration reached the goal state. It was much slower than VI and PI, requiring ~32 seconds to produce a good policy. The Mean Values by Episode curve in *Figure 5* shows an initial dip as the agent tries actions at random and learns the impact of those actions, before learning enough about the environment to start making better decisions. It's interesting to note that Mean Value was directly proportional to Epsilon Decay but Mean Cumulative Rewards were highest in more moderate values of the decay range. I believe this is because with very high decays, as actions are more random even in later episodes, the state values become oversaturated as the goal reward value spreads out across states that would not be chosen by a more informed (eg. less random) policy. For this reason, I chose to use mean Cumulative Rewards to pick Epsilon and Epsilon Decay and settled on 1 with decay of 0.99995 as highlighted in the table in *Figure 5*. The table suggests that decaying too quickly doesn't allow enough Exploration, while decaying too slowly prevents the learner from using what it's learned.

epsilon_decay	0.990000	0.995000	0.999000	0.999500	0.999900	0.999950	0.999990	0.999995	0.999999
epsilon									
1	2.99	2.47	2.98	3.43	3.31	3.86	3.31	4.12	3.82

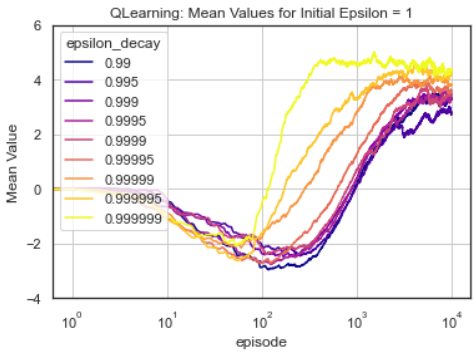


***Figure 5**: Q-Learning performance by Epsilon and Epsilon Decay. Table shows Mean Cumulative Reward.*

Figure 6 shows the nice balance between Exploration and Exploitation these values produced: the values initially suffer as the learner randomly tests actions, but start improving quickly as epsilon falls below 0.5 (where a majority of the actions are dictated by the policy). As the learner models the values through more episodes, the # of iterations required to reach the goal state decreases. In addition to the gradual Epsilon decay, the episodes themselves appear crucial. It took thousands of these episodes, with the agent starting in a random position, for the learner to converge. This captures the element of Random Restarts for exploration that we saw previously in the Randomized Optimization portion of the class.

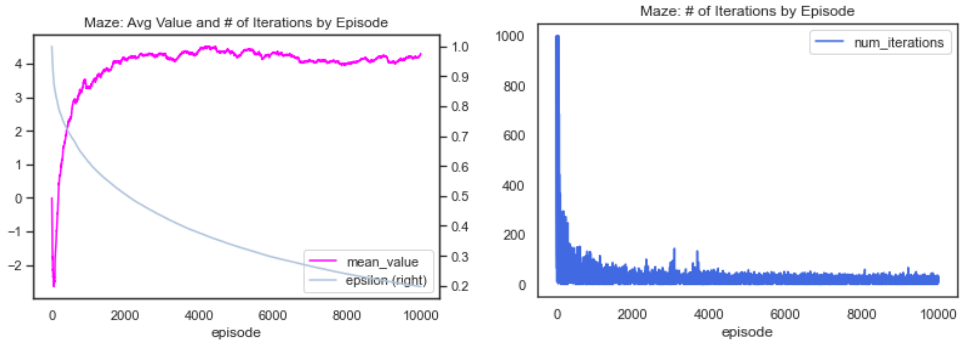


***Figure 6**: Q-Learning Mean Value and Epsilon by Episode, final policy, and # of Iterations by Episode. Gamma = .99, Epsilon = 1, Epsilon Decay = 0.999995, apha = .2, no alpha decay.*

The policy produced is similar to that of Value Iteration and Policy Iteration at the same gamma = .99, but it presents a confused policy in rows 1 and 13 near the pitfall ring, with some states proceeding along the wall and others steering away from it. This explains why the Mean Cumulative Reward of 4.12 is slightly lower than that of PI and VI. With the simulated nature of Q-Learning, these states were probably not visited enough times on popular routes by the agent to catch many of the stochastic movement mistakes that were better modeled by VI and PI. At gamma of 0.8, the policy is more risk-averse: it avoids the risky shortcut from [7,1] to [7,2], and travels around the -7 reward wall from [7,8] instead of going through it. As discussed earlier, the heavier discount diminishes the value of the risky shortcut paths as the greater near term risk outweighs the more discounted later reward.

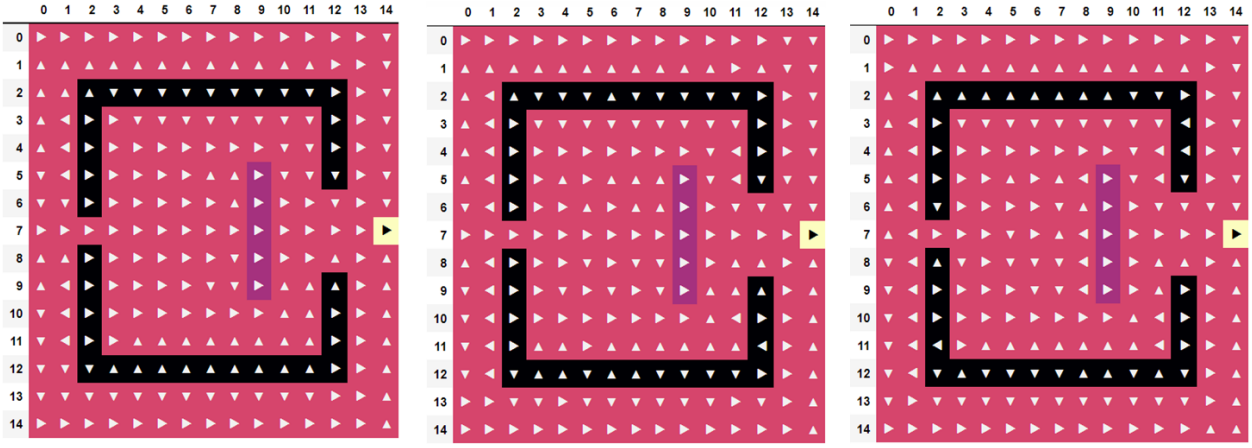


***Figure 7**: Value Iteration and Policy Iteration policy (left) with gamma = .99, Q-Learning policy with gamma = .99 (center), Q-Learning policy with gamma = .8 (right).*

Overall, Q-Learning produced a final policy that led to Mean Cumulative Reward of 4.12, which is slightly lower than those obtained by VI and PI. More random start episodes might improve this, but will be expensive. It produced a Mean Value of 4.29 which was similar to that of VI and PI. It takes a lot of Episodes to achieve this, and is more costly computationally. For this problem, if the transitions are able to be properly modeled by a known Transitions and Rewards matrix for each state and action, VI and PI appear to be a faster route to finding the optimal policy.

Forest Management - Value Iteration and Policy Iteration

I ran Value Iteration and Policy Iteration on the Forest Management problem. Convergence and iterations were determined in the same manner described for the Maze problem, except without stopping in a terminal state as there is no terminal state. Both approaches produced identical policies that don't cut (policy action = 0) in the first state, cut (policy action = 1) in states 1-653, and don't cut in states 654-1,000. This is intuitive: if you inherit a forest that's in its 10th year of life and there's a .001 probability of a fire each year, the cumulative probability of a fire occurring by year 1,000 surpasses .5 near the 500th year, so it's best to simply harvest the forest now vs. waiting on an unlikely large reward at year 1,000 given the risk of a fire destroying the forest in the future. But, what if the forest is in its 800th year of life? At that point, there's only a 0.2 probability that it will burn in the next 200 years, so it's wise to wait for the larger reward in the last state.

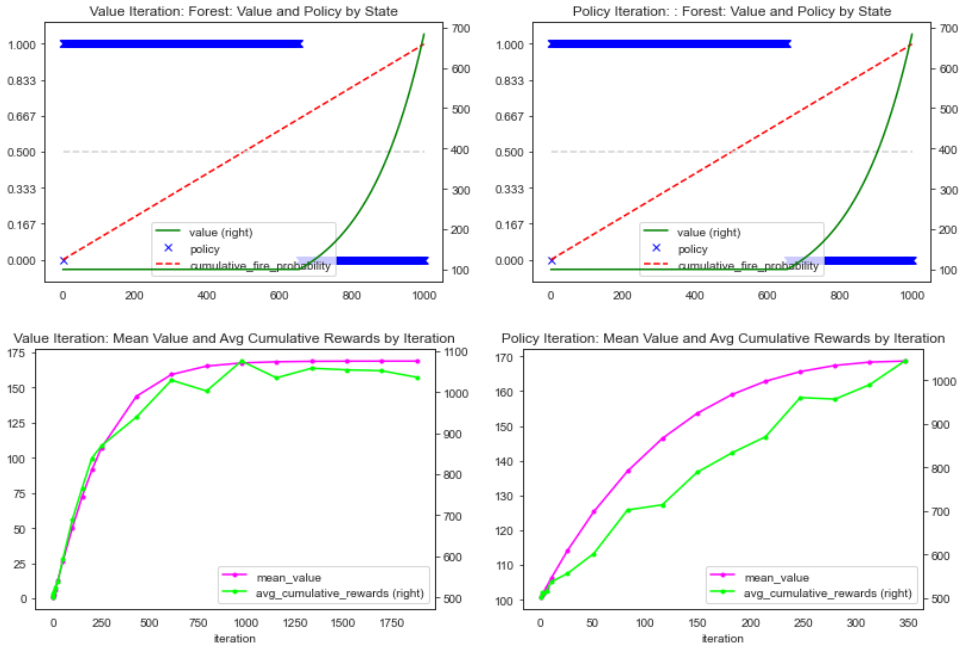
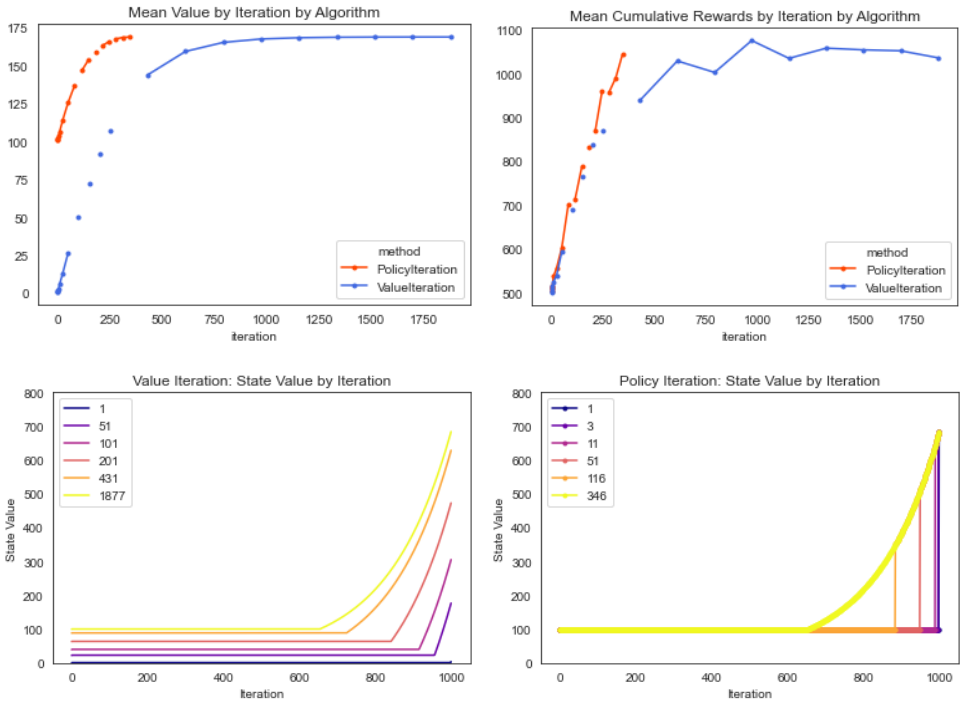


***Figure 8**: Value Iteration and Policy Iteration for Forest Management. Top: Value, Policy and Cumulative Fire Probability by State. Bottom: Mean Value and Mean Cumulative Rewards by Iteration by Algorithm. Policy 0 = wait (don't harvest).*

So why doesn't the policy suggest cutting at the midpoint of the states? It's because the discount (I used 0.995) diminishes the final state's reward, so it's only worth it to wait in the states that are even closer to the final state. Using a lower discount of 0.9, Value Iteration produced a policy that was even more conservative: cut was recommended up until the 980th state. This is similar to the behavior we saw in the Maze problem: the policy was more conservative around riskier obstacles in the maze the lower the

discount became, suggesting near-term rewards outweighed obtaining a highly discounted but originally larger reward many states into the future.

Both algorithms produced similar Mean Cumulative Rewards (simulated using 2k episodes each with a max of 1k iterations, starting in random states) : 1035 for VI and 1044 for PI. Value Iteration took 0.5 seconds and 1,877 iterations, while Policy Iteration took 9.3 seconds and 346 iterations. Searching in Policy vs. Values Space requires less iterations, but more work to obtain an improved policy in each iteration. Unlike the smaller Maze problem, Policy Iteration's Mean Cumulative Rewards started out closer to that of Value Iteration in the first iterations (see the top plots in *Figure 9*). With more states and a fleeting probability of reaching the final state's reward, the initial locally-informed policy in Policy Iteration didn't prove as useful. Policy Iteration also improved linearly, needing the last few iterations to reach a decent Mean Cumulative Rewards (see the bottom of *Figure 8)*. This is in contrast to Value Iteration where most of the improvement occurred in the first 20% of iterations. Examining the bottom of *Figure 9*, we see that the last state's reward cascades across the nearby states earlier on in Value Iteration than Policy Iteration. This was the opposite of what we saw in the Maze problem. In Forest Management, with so many states dictating an obvious cut policy, iterating in policy space vs. value space is not likely to produce material differences until the algorithm mostly models the earlier states' cut policy and picks up on the last state's reward for waiting. This is demonstrated in the bottom of *Figure 9.* This contrasts with Maze where there is more variation in policy as Policy Iteration progresses.



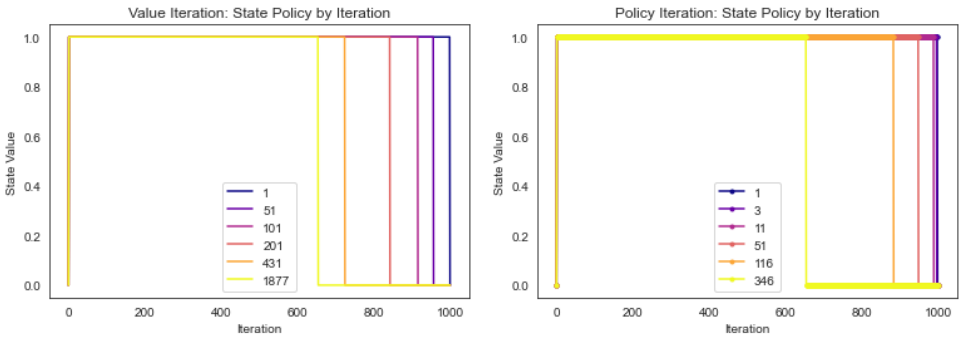


***Figure 9**: Value Iteration and Policy Iteration for Forest Management analysis by Iteration. Top: Mean Value and Mean Cumulative Rewards by Iteration by Algorithm. Middle: State Value by Iteration by Algorithm. Bottom: State Policy by Iteration by Algorithm.*

Forest Management - Q-Learning

I ran Q-Learning for the 1k state Forest Scenario problem using 10k episodes and 1k max iterations per episode. Immediately apparent in *Figure 10* is Q-Learning's struggle to correctly learn this scenario in a reasonable # of episodes. Unlike VI and PI, Q-Learning learns by randomly testing actions and remembering the results. Unless enough actions from the penultimate state to the ultimate state are learned, and then actions from the preceding adjacent states from the penultimate state are learned and so on, Q-learning will have a hard time modeling the conservation reward that only exists in the final state. For this reason, Q-Learning is hampered in that its ability to Explore (with one particular important state transition being crucial) is severely diminished. In this scenario, the cut action that is popular for the earlier states sends the agent to state 0, at which point a cut is performed at state 1 and the cycle is repeated. One could imagine modifying the algorithm to work backwards from the final state to help it along, but I feel like this would interfere with the "Model Free" nature of Q-Learning. Doing this would bring the approach closer to that of VI and PI, which are already established and performed well on the problem as demonstrated previously.

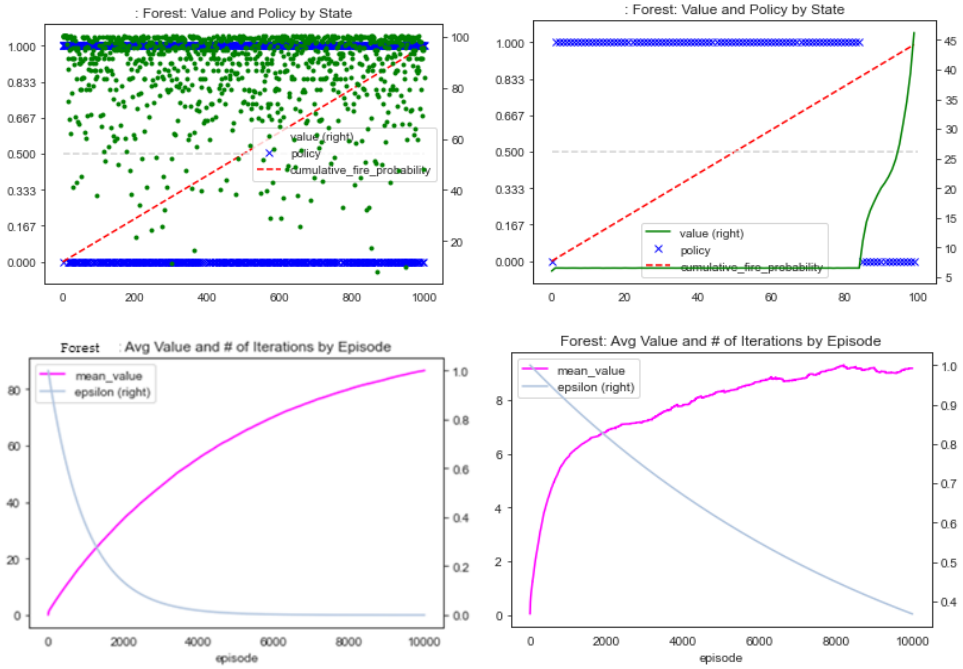


***Figure 10**: Q-Learning Results for the Forest Management problem. Left: 1,000 states, 0.999999 epsilon decay. Right: 100 states, 0.999999 epsilon decay.*

Various values of slower and faster epsilon decay all led to a similar Mean Cumulative Reward of 500, which was half that of PI and VI and is basically a representation of cumulatively accrued cut/harvest actions over manys states. Trying more episodes was prohibitively expensive.

That being said, out of curiosity, I did try Q-Learning on this problem using a smaller problem size: 100 states, 1/100 probability of fire, 100 max iterations, 1k episodes at 0.92 decay (adjusted down for less states to behave similarly to the 1,000 state scenario). The structure of the problem is similar at this size, and is more approachable for the random exploration episodes Q-Learning requires. At an epsilon decay of 0.999999, the learner was able to reach a Mean Cumulative Reward of 84 within 50 seconds. For a benchmark, Value Iteration produced a Mean Cumulative Reward of 92.4 on the same problem. This epsilon decay is very slow and Epsilon was still 0.4 by the 10,00th episode. This contrasts with the Maze problem, where more moderate decays performed best (eg. epsilon had reached 0.2 by the final episode). Although random action rate was still high by the end of the episodes, the learner still modeled the reward actions and saw values increase toward the final higher award state. This means that the learner benefitted from more random actions to happen upon sequences of "wait" actions in the later states (even when the policy at that time was "cut") that led it to the final conservation reward, allowing that value to cascade back into the preceding states. While it still modeled the later-state values adequately enough to produce a decent policy, the high epsilon suggests that the algorithm was doing something more similar to Randomized Hill Climbing vs. performing the Exploration/Exploitation balance that Q-Learning should perform, which explains why this problem poses a particular challenge for Q-Learning at 1,000 states.

	epsilon decay								
epsilon	**0.99990000**	**0.99995000**	**0.99999000**	**0.99999500**	**0.99999900**	**0.99999950**	**0.99999990**	**0.99999995**	**0.99999999**
1	53.495	53.07	54.675	71.085	84.065	82.865	83.74	80.575	83.34

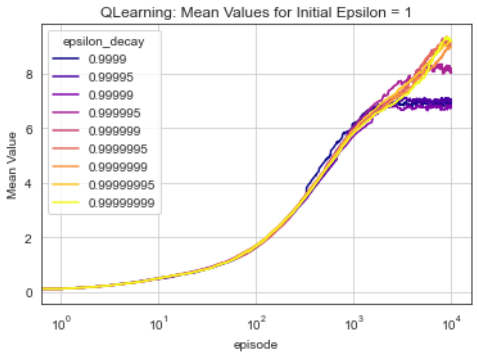


***Figure 10**: Q-Learning Results for the Forest Management problem at 100 states by Epsilon decay. Table shows Mean Cumulative Reward. Value Iteration reached 92.4 Mean Cumulative Reward on the same problem.*

Overall, for this problem, I would explore using another Reinforcement Learning algorithm for a more efficient learning process on this Forest Management problem at sizes greater than a couple hundred states.