Markov Decision Processes

Blake Wang
bwang400@gatech.edu

1 Problems

1.1 Forest Management

The first Markov Decision Process (MDP) I chose is Forest Management. In this problem, there is a forest which could burn down any time with probability p. Each year, the possible actions are “wait” and “cut”. The primary goal is to preserve the forest and the secondary goal is to profit by cutting down the forest for wood. I chose to use MDP Toolbox for Python’s forest module which implements this problem with the following parameters.

- S: number of states
- p: probability that the forest will burn down
- r1: reward for performing “wait” while at the last state
- r2: reward for performing “cut” while at the last state

This problem is interesting because it has two competing goals in an uncertain environment. The last (oldest) state is special in that the rewards differ from the rest.

1.2 Frozen Lake

Second, I chose the Frozen Lake problem. In this problem, the agent must walk across a square grid from a start location to a frisbee. Each cell of the grid is either the starting point, a frozen section, a hole in the ice, or the frisbee (goal). The agent can perform one of four possible actions; “up”, “down”, “left”, and “right” to get to the frisbee however the result of these actions are probabilistic. In other words, the lake is slippery. This problem is interesting because of its stochasticity. Because actions are not deterministic, this affects the optimal policy. For example, sometimes the optimal policy may dictate that the agent moves farther away from the frisbee in order to avoid a hole in the ice. I chose to use the FrozenLake-v0 implementation of this problem by OpenAI Gym. This implementation supports custom maps which gave me the flexibility to play around a bit. In this formulation of the problem, the frisbee cell has reward +1, holes in the ice have reward -0.75, and all other cells have reward of 0.

2 Value Iteration and Policy Iteration

2.1 Varying Problem Size

I started with the Forest Management problem. First, I played around with the problem parameters and observed their effects on Value Iteration (VI) and Policy Iteration (PI). As a first experiment, I varied problem size by increasing the number of states S while fixing the remaining problem parameters to their default values (r1=4, r2=2, and p=0.1). I set the discount value to 0.95.

Figure 1 - Comparison of VI and PI with respect to Forest Management problem size

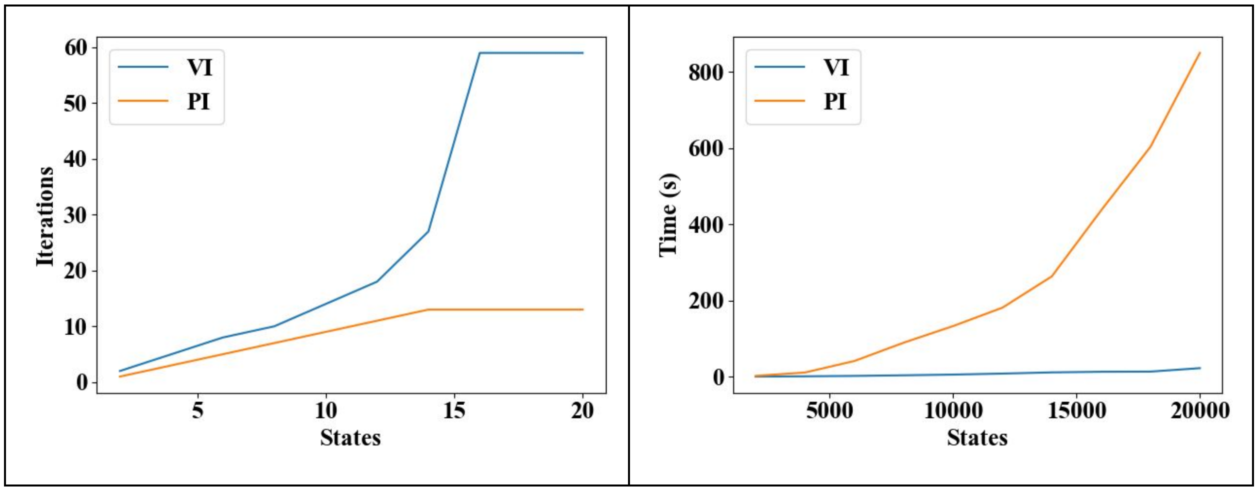


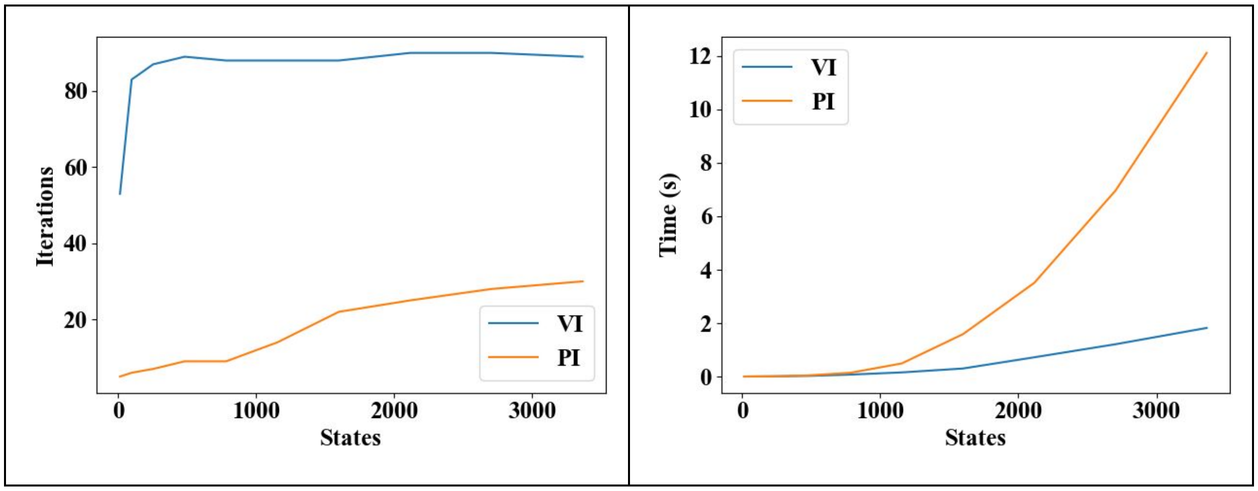
Figure 1 shows how varying S in this problem affects iterations (left plot) and time (right plot) for VI and PI. We can see that, as expected, PI takes fewer iterations to converge at every problem size. This is because a policy will usually stop changing between iterations (converge) sooner than the value function itself converges. Of course, setting ε high enough would make it so VI converged first, though in this case we risk converging too soon and not returning an optimal policy.

It is interesting that the VI iterations curve suddenly flattens at 59. Upon seeing this, I was initially so skeptical that I modified the library code to hardcode a large max_iter value because I did not trust that the value I supplied was being used. This had no effect. It turns out that with these problem parameters, VI will converge in 59 iterations no matter how much larger the problem gets. PI also has a maximum number of iterations at only 13 but because the transition was more gradual, it did not cause me such alarm.

It is also interesting that though VI takes more iterations to converge, it takes less time. This difference is magnified as problem size increases. Note that the time comparison plot in FIgure 1 shows problem sizes up to 20,000 states though number of iterations has been constant across almost all of those values. So even though both of the iteration curves are totally flat, larger problems have greater time per iteration, especially for PI. This is because the cost of matrix multiplications performed by PI at each iteration grows with the number of states. Lastly, I will note that for every problem size I tried, VI and PI converged on the same optimal policy.

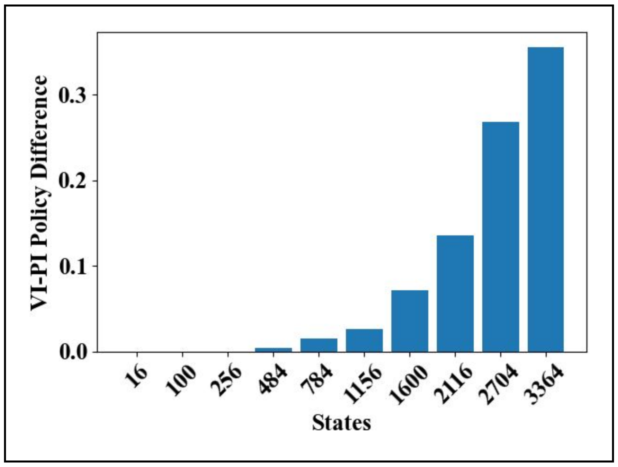
Next, I conducted the same problem size experiment with Frozen Lake. I wrote a map generator that took in the edge length of the (square) lake as input and then generated maps with a deterministic hole pattern across the lake. I leveraged FrozenLake-v0's desc parameter to feed in these custom maps and generate OpenAI Gym environments. I then converted the environment into transition and reward matrices formatted for MDP Toolbox. This allowed me to use the same ValueIteration and PolicyIteration implementations as before for the sake of comparison.

Figure 2 - Comparison of VI and PI with respect to Frozen Lake problem size



We can see in Figure 2 that by meddling with various custom map hole patterns, I was able to make the problem a bit more interesting. These plots further demonstrate the differences between VI and PI. Again, VI converges in many more iterations but in much less time. The right-hand plot of FIgure 2 shows that for problem sizes above ~1000 states, the gap in convergence time between the two algorithms increases rapidly. VI is clearly a better choice for this problem at large sizes, despite the much higher number of iterations performed.

Figure 3 - Percentage difference between VI and PI policies as problem size increases



Lastly, we can examine in Figure 3 the degree to which the policies generated by VI and PI disagreed for the various problem sizes. For the first three problem sizes, the policies were identical. However, when I used a 22x22 grid (484 states) the policies began to recommend different actions in certain states. Figure 3 shows that the larger the grid world, the more the policies diverged. The largest grid world (3,364 states) yields a 35% disagreement. So which algorithm's policy is closer to optimal? At this point I haven't yet done any hyper-parameter tuning, so it is not very meaningful to say. I'll do that next.

2.2 Hyper-parameter Tuning

After doing the size-varying experiment, I settled on the choice of a 30x30 Frozen Lake (900 states) as my "large" problem and Forest Management with only 15 states as my "small" problem because I think these will be the most illuminating. Though 900 states is not incredibly high, it clearly demonstrates the differences between VI and PI. The large problem yields very different performance and policies between VI and PI and the small problem yields very similar performance and identical policies. So, with the problems fixed, I started to tune the algorithms.

Figure 4 - Mean long term reward over multiple walks as a function of discount

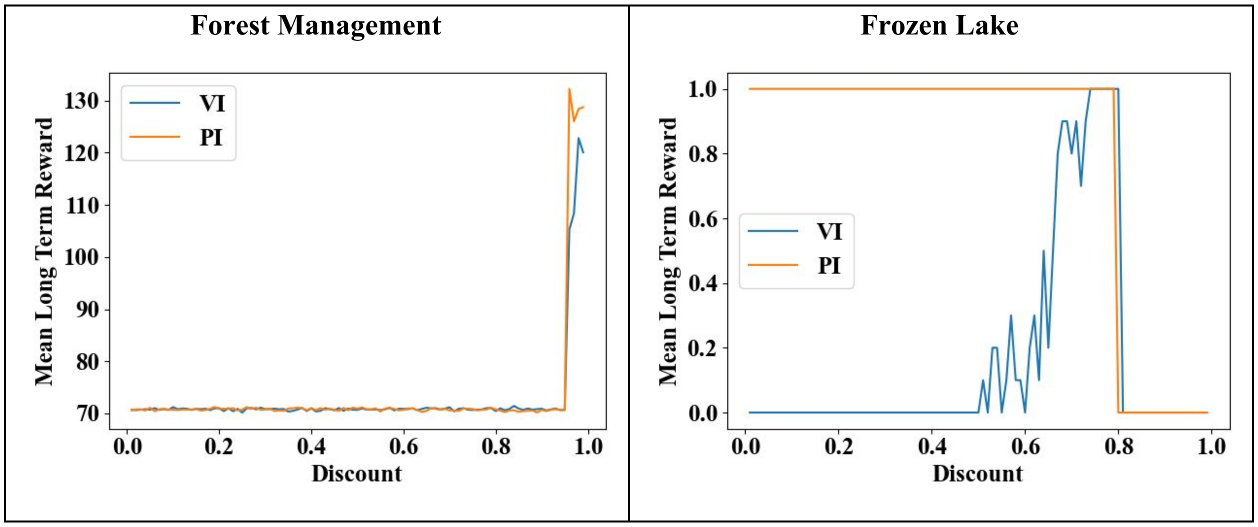


Figure 4 shows how mean long term reward varied in each problem as a function of discount. I calculated these values by executing a number of "walks" for each in a range of discount values and taking the mean reward earned over all the walks. By walk, I mean a series of steps in which the agent starts at the initial state then acts according to the policies prescribed by VI and PI, collecting reward along the way. The agent stops when one of the following conditions is met.

- The Frozen Lake agent falls into a hole in the ice
- The Frozen Lake agent finds the frisbee
- Either agent performs the maximum number of steps (set to 10x number of states)

The (small) Forest Management problem (left side) shows that for most discount values the mean long term reward is near constant (~70) then shoots up to ~130 at discount values near 1 for both algorithms. Specifically, I found that the best discounts for VI and PI are 0.98 and 0.96 respectively. Using these discount values nearly doubled the mean long term reward as compared with most of the other values. I also tried various values for ε for VI however they had no effect whatsoever presumably because of the tiny size of the problem.

The results from the (large) Frozen Lake experiment (right side) were quite different. Both curves have sections which stay constant at 0.0. I supposed that this was probably due to the agent walking around in cycles, neither getting closer to the frisbee nor falling into a hole in the ice. The maximum steps value was very high which supported this theory. When I investigated further by examining the actual moves in each walk, I found this to be true. In a grid world where each step had some small negative reward instead of 0, these cycles would probably not have occurred as frequently because the agent would've been in more of a hurry. We can see that VI was very sensitive to discount, earning the optimal mean long term reward of 1.0 when the discount was 0.74. The noisiness of this plot might have been decreased somewhat if I had taken the mean over even more walks. Conversely, PI does not seem to care at all about discount value up until 0.8. Before this point, PI is able to converge on an optimal policy every time but after this point, reward falls down to 0.0 (agent walking in cycles). It makes sense that PI would be less sensitive to tuning because each iteration only remembers the policy not the value. I'll also mention that I had to make ε very small, 10^{-7}, in order to get any reasonable policies from VI. To find this value I searched over powers of 10.

2.3 Convergence

Now with the problem sizes and algorithm hyper-parameters tuned to maximize interesting-ness, we can look at the actual convergence of VI and PI on these datasets. The fairest way I could think of was to compare mean long term reward (mean over a number of walks, similar to the previous section) as the algorithms converge. I performed the following.

- Execute a single run each of VI and PI, writing down the timestamp and the policy at each iteration
- After the runs completed, perform a number of walks using each of those policies to find the mean long term reward
- Plot these mean long term reward values against their respective timestamp

Figure 5 - Mean long term Reward over time

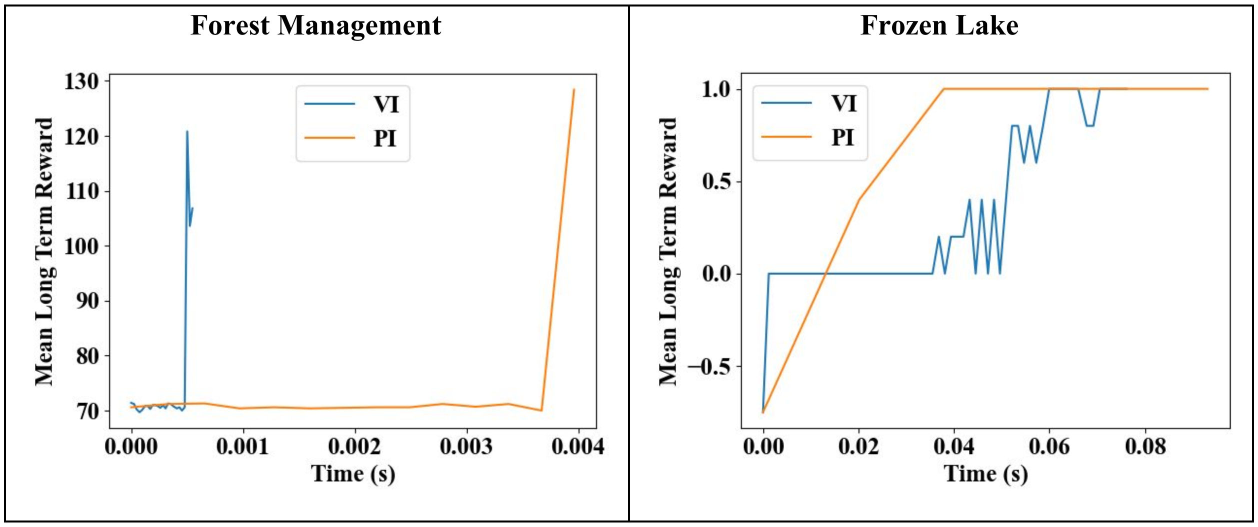
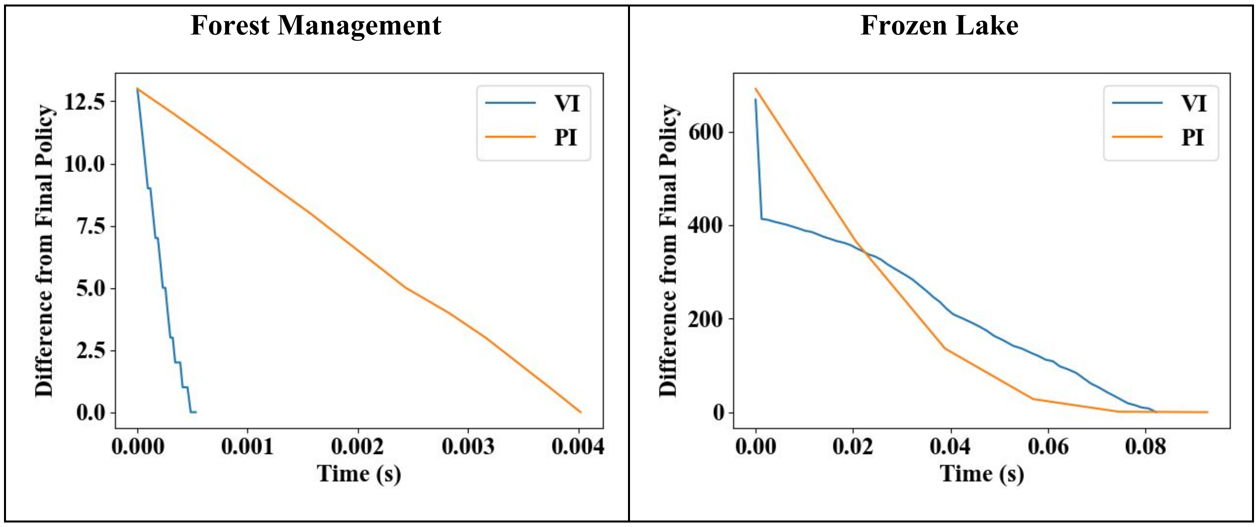


Figure 5 shows the results of this experiment. Let us look at the Forest Management (left side) plot first. The two curves behave very similarly to each other however PI "waits" a lot longer before suddenly shooting up. This is consistent with the rest of the experiments in which PI takes longer to converge, at least as measured in wall clock time. Another interesting observation is that the VI curve does not finish on the policy with the highest mean long term reward. It dips down a little at the very end. This could just be noise and if I averaged over a greater number of walks, this dip might vanish.

The Frozen Lake plot (right side) reveals something even stranger. PI actually converged on an optimal policy before VI (~0.04s), however it continued to spin until after VI completed, which made it *appear* to have slower convergence. Again, I could increase the number of walks and I might observe that the iterations between ~0.04s and ~1.0s were actually making some small improvement. We can also see that again, the PI plot is monotonic in mean long term reward while the VI plot is not. In other words, any given PI iteration improves mean long term reward, but this is not the case for VI.

Figure 6 - Difference from final policy over time



The policies and their timestamps which I wrote down perform walks for Figure 5 yielded one more set of interesting plots. Figure 6 shows how different the current policy was from the final policy at every timestamp as measured by the number of states for which the current policy recommended a different action. The Forest Management plot (left side) shows that as time progressed, the policy approached the final policy linearly for both algorithms.

The Frozen Lake plot (right side) shows that VI took a giant leap towards the final policy at the very beginning then approached it roughly linearly afterward. The PI plot on the other hand starts off looking fairly linear then slows its approach toward zero. Interestingly, the point where it begins to slow (~0.04s) is the same point which Figure 5 showed us that PI had already reached optimal mean long term reward.

3 Q-learning

In this section, I'll describe experiments with Q-learning (QL) to solve the MDPs described in sections 1 and 2. The major distinction between Q-learning vs VI and PI is that with Q-learning we don't know the model ahead of time. Because of this, we must interact with the environment in order to learn. This begs the question of what is the best way to interact with the environment. In other words, what is the best exploration strategy? This section attempts to answer that question for each problem.

3.1 Hyper-parameter Tuning

I first set out to tune the discount value for the Forest Management problem. I used the same method as n section 2.2, shown in Figure 4. I varied the discount value passed into QL and then used the resultant policies to guide "walks" over the states in order to see which discount yielded the highest mean long term reward. At first, I could not get any values for mean long term reward that were near those of VI and PI. I tried increasing the QL iterations to very high (10^7) but this did not help. I then tried switching up the exploration strategy. MDP Toolbox's QLearning implementation uses decaying ε-greedy exploration. So after this approach failed to yield good results I tried Upper Confidence Bound (UCB) exploration. I modified the QLearning implementation for UCB and ran the same experiment varying the discount value and keeping the number of iterations very high, but *still* did not get good results. I tried different values for β, the parameter for determining how explorative vs exploitative to behave and this did not help either.

The experiment with UCB did have an upside however. UCB tracks how many times a given action has been explored in each state. Inspecting this matrix revealed that first and last states were receiving thousands of iterations while many states were receiving only a few, even with high β. Inspired by this observation, I decided to try pure random exploration where every iteration chooses both a random state and a random action to explore. This finally worked! With a roughly even distribution of knowledge across all states and actions, QL was able to converge on the same optimal policy as VI and PI did. I was then able to proceed with finding a good discount value. I then tuned the number of iterations by holding this discount value fixed and varying iterations.

Figure 7 - Hyper-parameter tuning QL with random exploration for Forest Management problem

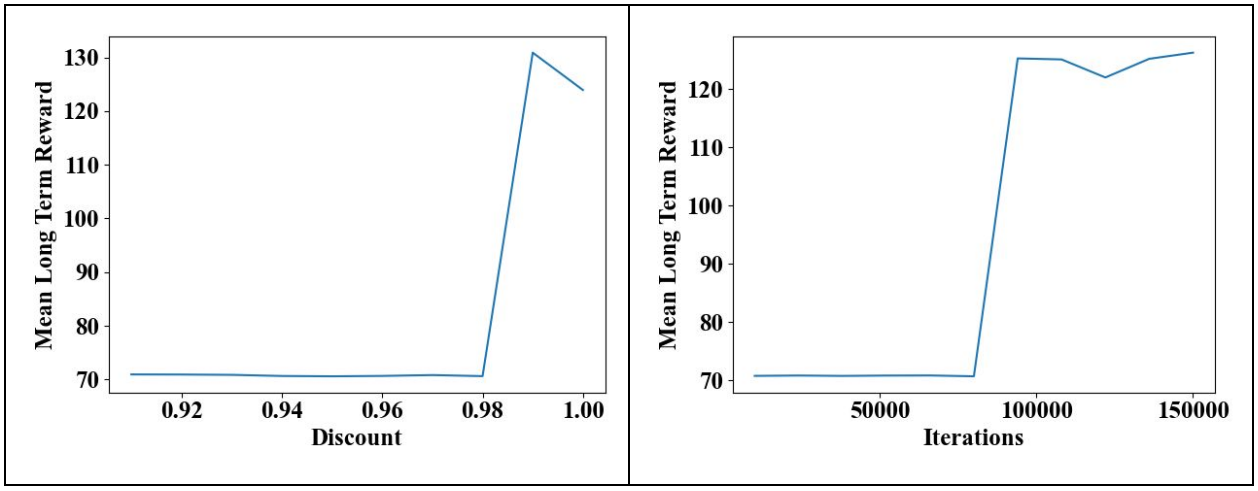
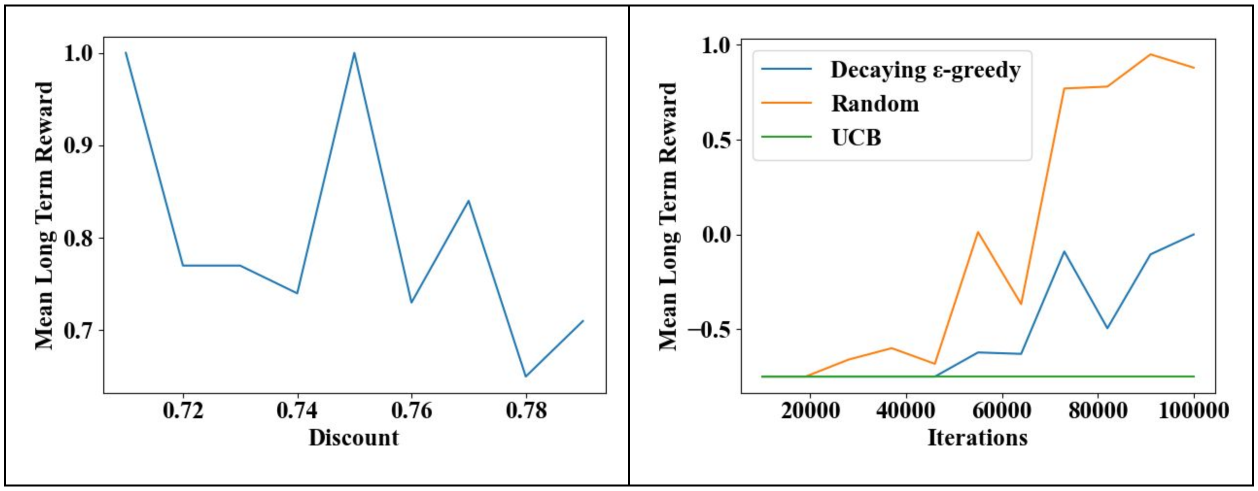


Figure 7 shows that the best discount value for QL with random exploration is ~0.99 and that the minimum number of iterations for optimal policy convergence is ~94,000. The variation at after that point is purely noise in the mean long term reward calculation. I verified this by checking to see that the policies after this point all agreed with each other perfectly. I then performed the same tuning of discount and number of iterations for the Frozen Lake problem.

Figure 8 - Hyper-parameter tuning QL for Frozen Lake problem



The left plot in FIgure 8 shows the results of varying discount while holding iterations fixed at a 10^6 (larger than necessary). For this experiment, I set QL to use random exploration as before. We can see that 0.75 yielded an optimal policy. On the right side of Figure 8, we see how the three different exploration strategies behaved while varying the number of iterations. Once again, randomly searching the space is the clear winner. It is the only one that converges on an optimal policy within 10^5 iterations. In contrast, UCB never even leaves the worst case (falling into a hole in the ice on each and every walk). Though not shown in this plot, I tried ε-greedy and UCB exploration at very high numbers of iterations (e.g. 10^7) and neither converged on optimal policies. This may be because the number of iterations is far greater than the search space of state-action pairs. If such a condition is true, uniformly distributing the exploration, and thus the learning, seems to be the best approach. On the other hand, if the search space is much greater than the number of iterations, then I hypothesize that a “smarter” approach like ε-greedy or UCB exploration would win the day.

4 Conclusion

Table 1 - Comparison of algorithms based on iterations and time across both problems

	Forest Management (15 states)		**Frozen Lake (900 states)**	
	Iterations	**Time (s)**	**Iterations**	**Time (s)**
VI	24	0.001	58	0.084
PI	14	0.010	6	0.144
QL (random explore)	94,000	2.334	96,000	26.48

With all the algorithms tuned and the data collected (represented in Table 1), we can now compare each algorithm in the context of both problems. As expected, QL takes many (orders of magnitude) more iterations and much more time to converge on good policies than do VI and PI. This is because QL must interact with the environment in order to learn rather than be handed the model up front. Because QL can learn through interacting with the environment though, one could argue that it is much more useful in the real world. It is often the case in real world problems that we are not handed the model and must learn about it through interaction.