Jacob Meyers
Assignment 4 - Markov Decision Processes
CS 4641
14 April 2019

Introduction

The purpose of this assignment was to explore the performance of three algorithms used to solve Markov Decision Processes. Two of the algorithms were offline planners: Value Iteration and Policy Iteration. The third was a model-free reinforcement learning algorithm: Q-learning. These algorithms were evaluated on their ability to efficiently solve two different Markov Decision Processes, which will be described below.

The Markov Decision Processes

The first Markov Decision Process I used is called Frozen Lake. In Frozen Lake, an agent tries to navigate a frozen lake from a start point in the upper left corner of a grid to a lower right goal point. There are holes throughout the grid. If the agent falls into one of the holes or reaches the goal state, the episode is over. Transitions from one point in the grid to the other are non-deterministic: the ice is slippery, so occasionally instead of moving to the intended square, the agent will instead move to the left or right of its intended direction, but never backwards. There is a reward of 1 for reaching the goal state, -1 for falling in a hole, and a reward of -0.1 otherwise. Two differently sized grids for this problem were used to explore how the number of states affected the performance of the algorithms : 8x8 and 20x20. I thought this problem was interesting because despite its relative simplicity, it has some interesting properties. At some level, this is a modified Gridworld: an agent navigates a grid, trying to reach a goal square at the other end of the map from where it starts. The state space will be relatively small. The states can be described uniquely with an x,y pair, and there are only four possible actions an agent can take - move up, move down, move left, or move right. This should allow the algorithms to converge quickly. There is intrigue introduced with the non-deterministic transitions and the presence of holes with negative rewards. The agent will be forced to consider whether it is better to take cautious paths avoiding holes when possible, possibly taking longer paths and decreasing the cumulative reward or to take riskier paths with a chance of reaching the goal state faster.

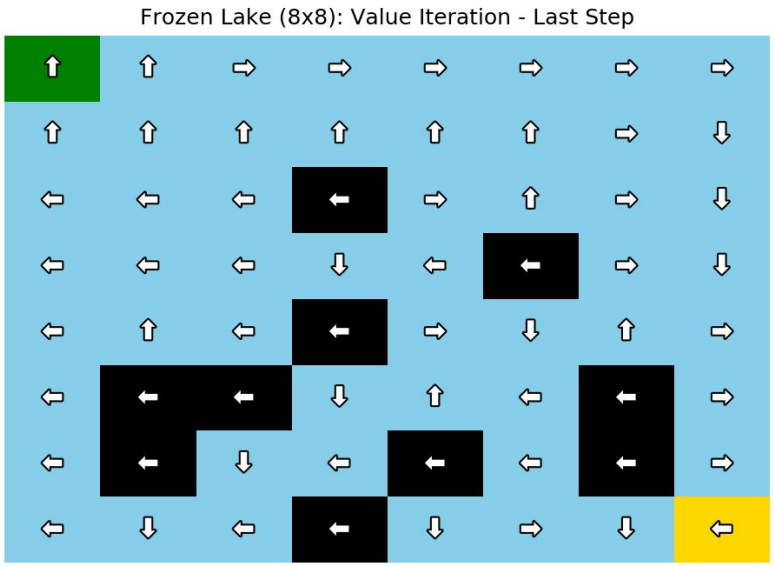
The second Markov Decision Process I used in is called Cliff Walking. It is also an adaptation of Gridworld, with an agent starting at the bottom-left of a narrow 4x12 map at the edge of a cliff, with the goal at the bottom-right on the other side of a cliff. There is a wind that sometimes blows the agent down, towards the cliff. The goal has a reward of 100. Stepping off of the cliff has a penalty of -100 and sends the agent back to the start. Otherwise, the agent incurs a penalty of -1. The reset mechanic is interesting to me. Since there is a huge penalty relative to the default reward of -1 and there still is the risk of being blown off repeatedly, incurring more and more -100 penalties. Since the world is relatively small in the y direction, the agent cannot really reach a safe distance from the cliff. This raises an interesting question: is it better to just dash across the row closest to the cliff, minimizing the time spent in front of the cliff or to play it safe and spend the extra time in front of the cliff to put some distance between the agent and the cliff? The non-determinism of the wind makes this a difficult question to answer. My intuition is that since the default reward is small relative to the reward of the goal and the penalty of falling off the cliff, the agent will be more inclined to take the longer route and put a buffer between itself and the cliff.

Methodology

There were three MDPs solved: 8x8 Frozen Lake, 20x20 Frozen Lake, and Cliff Walk. Each of these MDPs were solved by all three algorithms to compare results. Value Iteration and Policy Iteration each had a single parameter for which different values were tested - the discount factor (or gamma). The discount factor influences how much a planner takes into account short-term vs long-term rewards. A lower value will lead to a policy that places more emphasis on short term rewards and vice versa. Values from 0.0 to 0.9 inclusive were all tested. For the Q-learner, there were four parameters which were tested. The learning rates (or alphas) used were 0.1, 0.5, and 0.9. This value determines how much the agent will consider recently learning information as opposed to prior knowledge. A higher alpha will result in an agent relying heavier on recently learned information, while a lower alpha will result in an agent relying heavier on prior knowledge. Initial q-values were either chosen at random or initialized to zero. Initial epsilon values used were 0.1, 0.3, and 0.5. An epsilon greedy strategy is used, meaning the agent will choose the action that will maximize the q-value, with a probability 1 - epsilon of choosing a different, random action. A higher initial epsilon will result in the agent exploring more of the state space, and thus potentially learning more information. Finally, the same discount factors tested for Value Iteration and Policy Iteration were tested for Q-learning, and it serves the same purpose for a Q-learner. The optimal set of parameters for each algorithm was found based on highest mean reward gained by following the policy found by the algorithm, breaking ties if necessary with median and maximum reward. I'll examine the results of each MDP independently, comparing the performance of each algorithm on that particular MDP.

Results: Small Frozen Lake

The first MDP I'll examine is the 8x8 Frozen Lake. Let's see the policy and values that Value Iteration converged to in Figure 2.



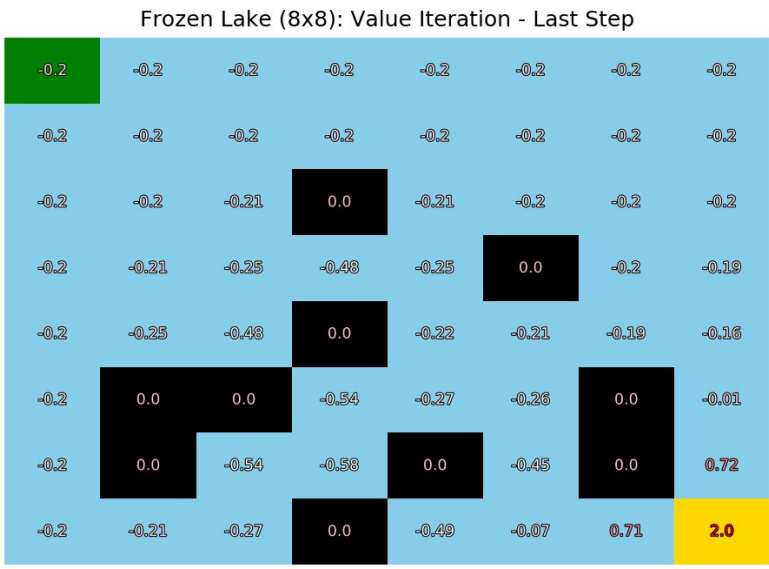


Figure 1. Final policy and values found by Value Iteration on 8x8 Frozen Lake with a discount factor of 0.5.

The first thing to note is how quickly Value Iteration converged, relative to both time and number of iterations. Value Iteration converged to the final value function in only 18 iterations. This can be explained by the relatively small state space of this particular MDP, as the state space is only 64. This could also explain why the mean reward was maximized by a somewhat lower discount factor - since the state space is not very large, the agent need not worry about too long term of rewards. An interesting feature of the policy shown above is the generally conservative nature of it - the agent is perfectly content to stay in the same state indefinitely, until the nondeterminism of the transitions results in it moving to a state it did not intend to. This policy, for the most part, makes little attempt at actually reaching the goal state. This is due to the reward structure of this particular MDP. Since the negative reward incurred at each time step (-0.1) is low relative to the negative reward incurred by falling in a hole (-1), the agent doesn't mind being conservative as long as it can avoid falling in the hole. You can see in the states bordering holes, the policy always immediately moves directly away from the hole. This way, there is no change of falling in the hole, since the agent will never move backwards from its intended direction. If the negative reward associated with each time step was higher, the negative reward associated with falling in the hole was lower, or the goal reward was much higher, then we would likely see a less conservative policy.

Let's now examine the Policy Iteration results in Figure 3, and compare them to the Value Iteration results.

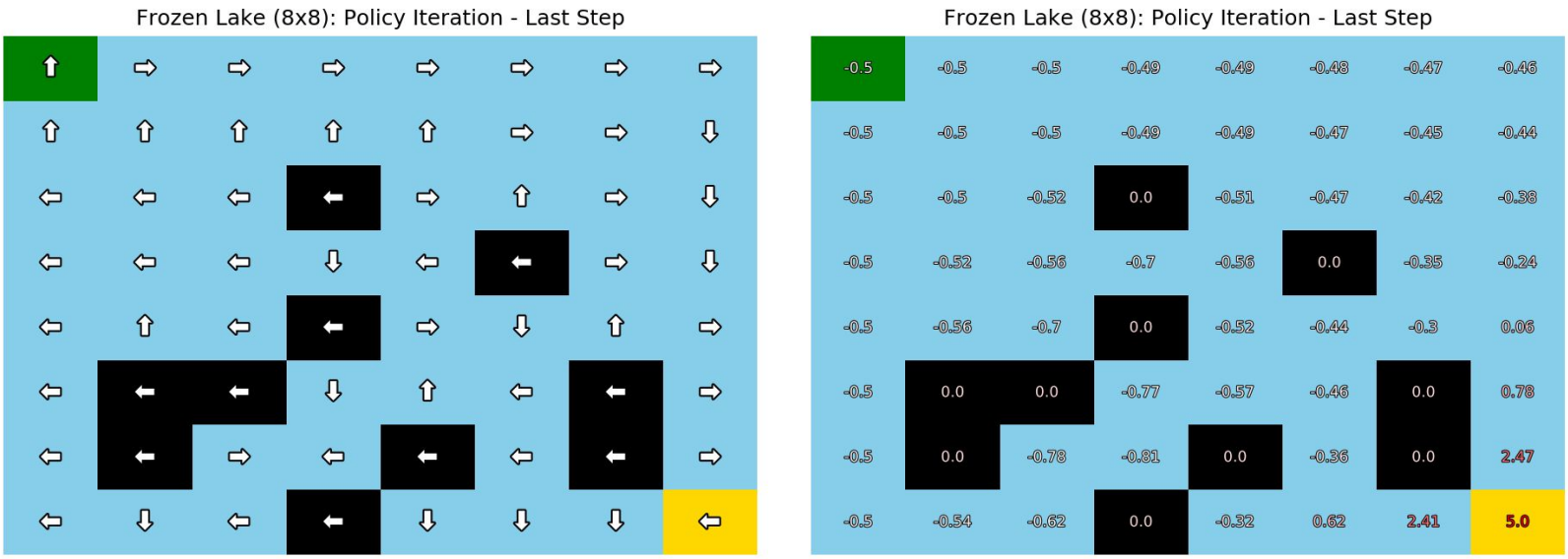


Figure 2. Final policy and values found by Policy Iteration on 8x8 Frozen Lake with a discount factor of 0.8.

For the most part, the final results are very similar. Since the discount factor used was 0.8, the magnitudes of both positive and negative rewards are larger, since the values of the states are more

heavily influenced by the values of its neighboring states. Also, since the discount factor is higher, the positive rewards propagate out from the goal state a bit farther than in Value Iteration for the same reason listed above. The major difference between the two is convergence time. Policy Iteration took only five iterations to converge to its solution! There was a slight caveat which we can see in Figure 4: an iteration of Policy Iteration took around an order of magnitude longer than an iteration of Value Iteration. This is because for each step taken of Policy Iteration, the current policy being evaluated is allowed to converge before the improvement step is taken. In Value Iteration, we simply iteratively update the value function of each state until convergence.

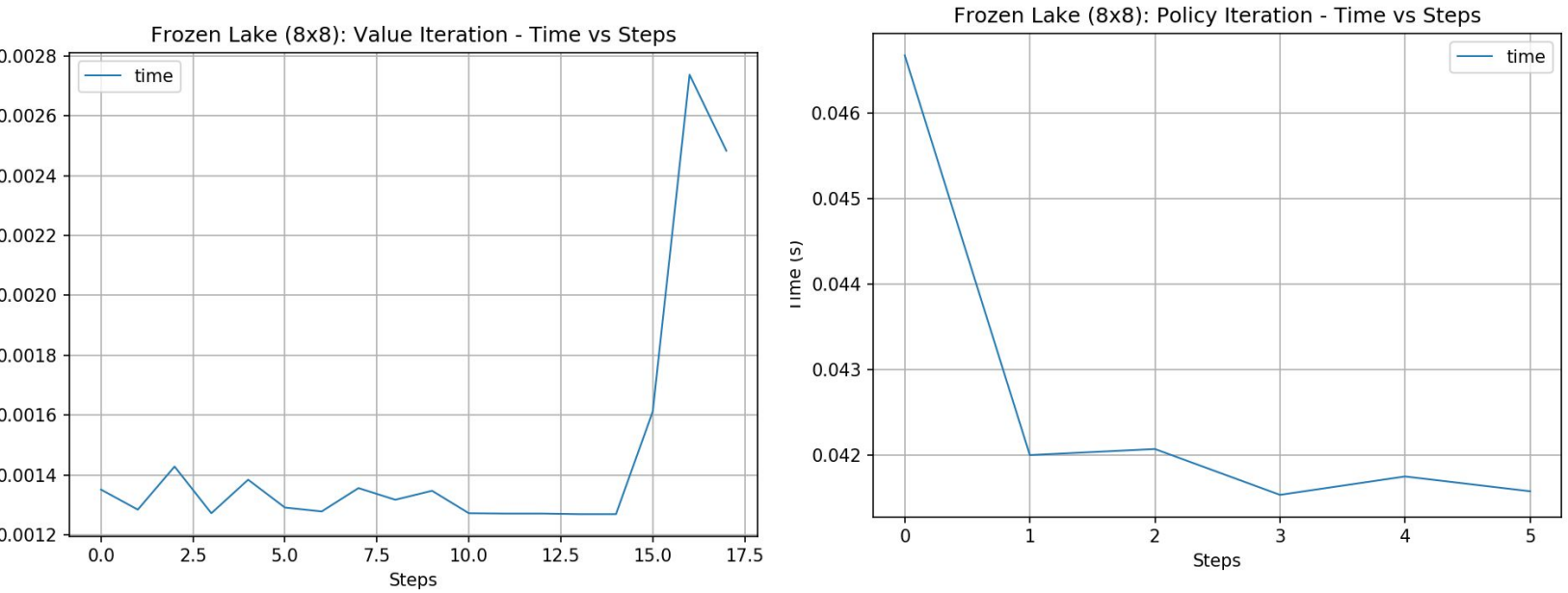


Figure 3. Comparison of time and steps taken by Value Iteration(left) and Policy Iteration(right) for 8x8 Frozen Lake. Time is in seconds.

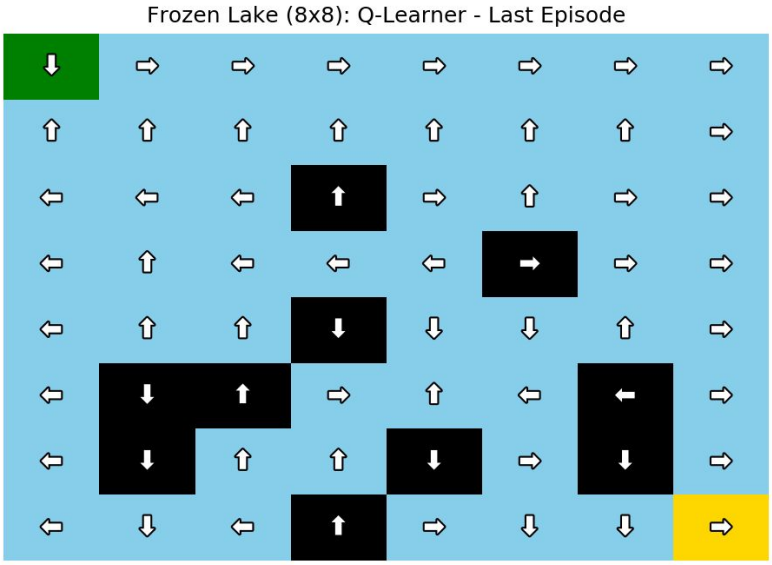
So, we may potentially be sacrificing a bit of performance with Policy Iteration. However, if we examine the absolute time taken by each algorithm, the difference is negligible for this MDP. There is an order of magnitude difference between the two, but this is a difference of a hundredth of a second versus a thousandth of a second. It will be more interesting to compare the time taken at each step for the larger Frozen Lake variant, since there will be a much larger number of states. If the difference in time between the two algorithms is non-trivial, we will more easily see it then.

The previous two algorithms had the advantage of full knowledge of the transition and reward model of Frozen Lake. Let's see how Q-learning performs, an algorithm which operates with no initial knowledge of the environment it is operating in.

At first glance at Figure 5, the policy looks quite a bit different than that of Policy Iteration and Value Iteration. The best-performing Q-learner was the one with an alpha of 0.5 and a discount factor of 0.1. This means the agent valued equally the However, this value was affected the least by its neighboring states, because of the very low discount factor. The end result is an agent that was the most willing to explore its environment due to the high alpha, but was nearly exclusively considering short-term rewards.

This could explain the generally conservative approach by the Q-learner in this environment, along with the magnitude of the rewards as discussed above.

So how did the Q-learner do, in comparison to the other algorithms? With respect to the policy found, fairly similarly. Q-learning had a better average reward, but the margin was only 0.002, while the time taken by Q-learning was significantly higher, due to the large number of episodes taken to learn the optimal policy. In Figure 6, we see that the time taken per episode is roughly on the same magnitude as each iteration of policy iteration. We also see that there were 1000 episodes taken by the Q-learner vs. 5 episodes taken by policy iteration. However, considering the fact that the Q-learning agent had no prior knowledge of the transition model or reward model, this extra overhead is to be expected. I would say that Q-learning performing on par (and even a bit better!) than the other two algorithms despite their advantage they hold with knowledge of the model is quite impressive.



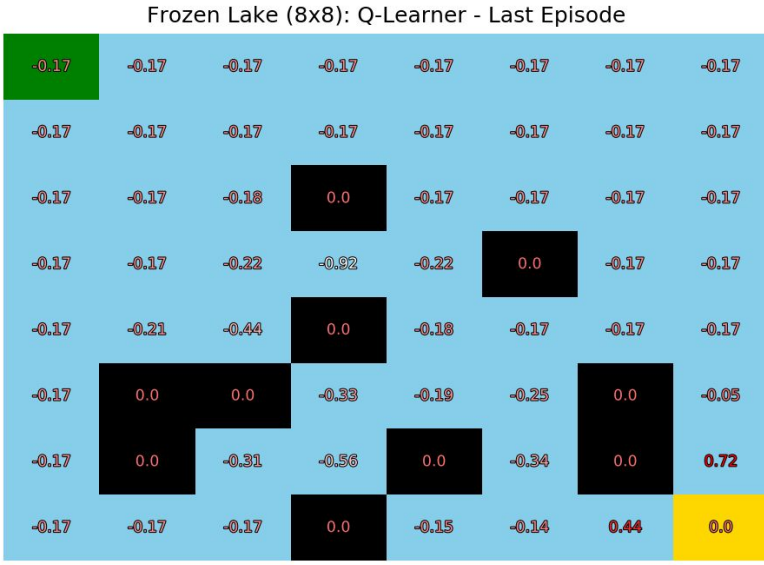


Figure 4. Final policy and values found by the Q-learning agent on 8x8 Frozen Lake with an alpha of 0.5, random initial Q values, an epsilon of 0.5, and a discount of 0.1.

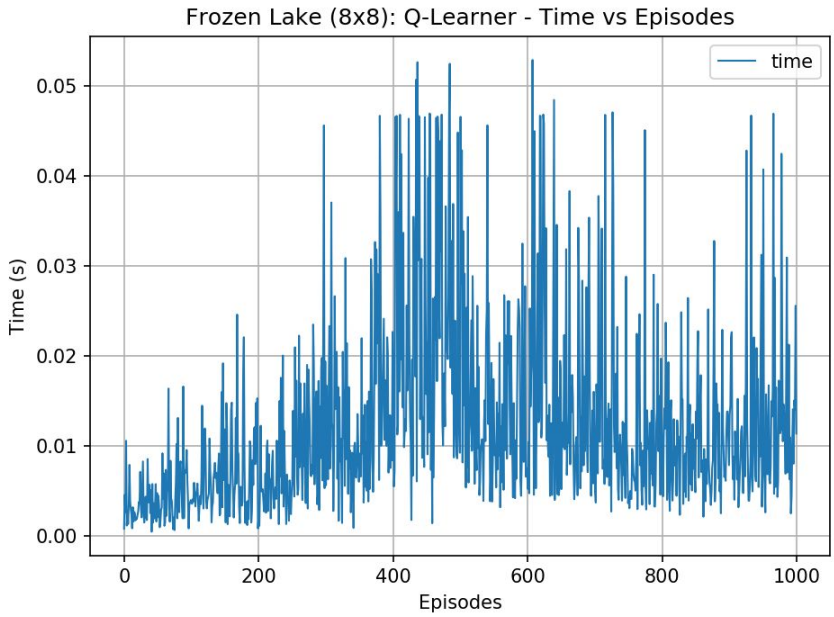


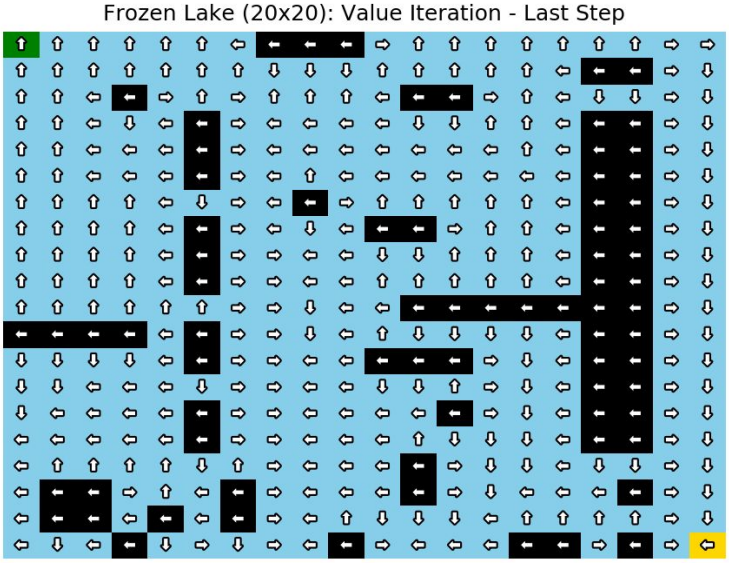
Figure 5. Time taken per episode by Q-learner on 8x8 Frozen Lake

Algorithm	Value Iteration	Policy Iteration	Q-learning
Mean Reward	-0.087	-0.087	-0.085

Figure 6. Average reward gained by each algorithm on 8x8 Frozen Lake.

Results: Large Frozen Lake

The next MDP I tested was the 20x20 version of Frozen Lake, which I will call Large Frozen Lake. The MDP is identical to the previously discussed one aside from the larger state space and additional hole squares. With this in mind, let's look at the results.



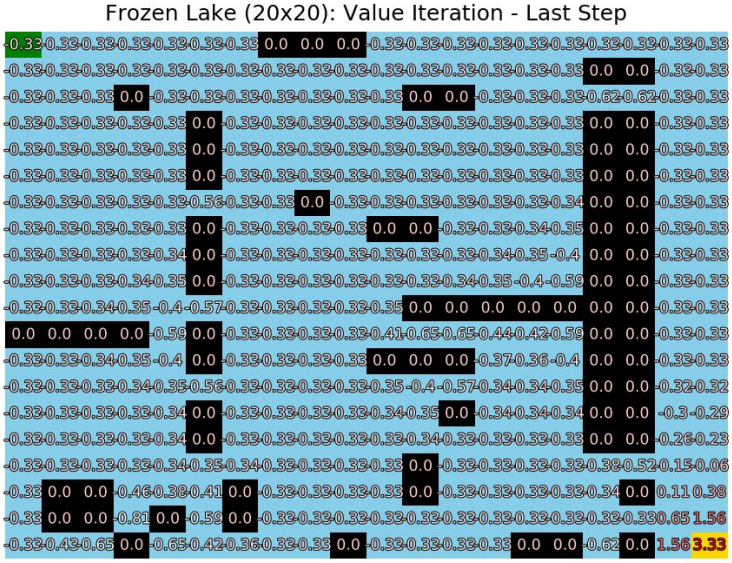


Figure 7. Final policy and values found by Value Iteration on 20x20 Frozen Lake with a discount factor of 0.7.

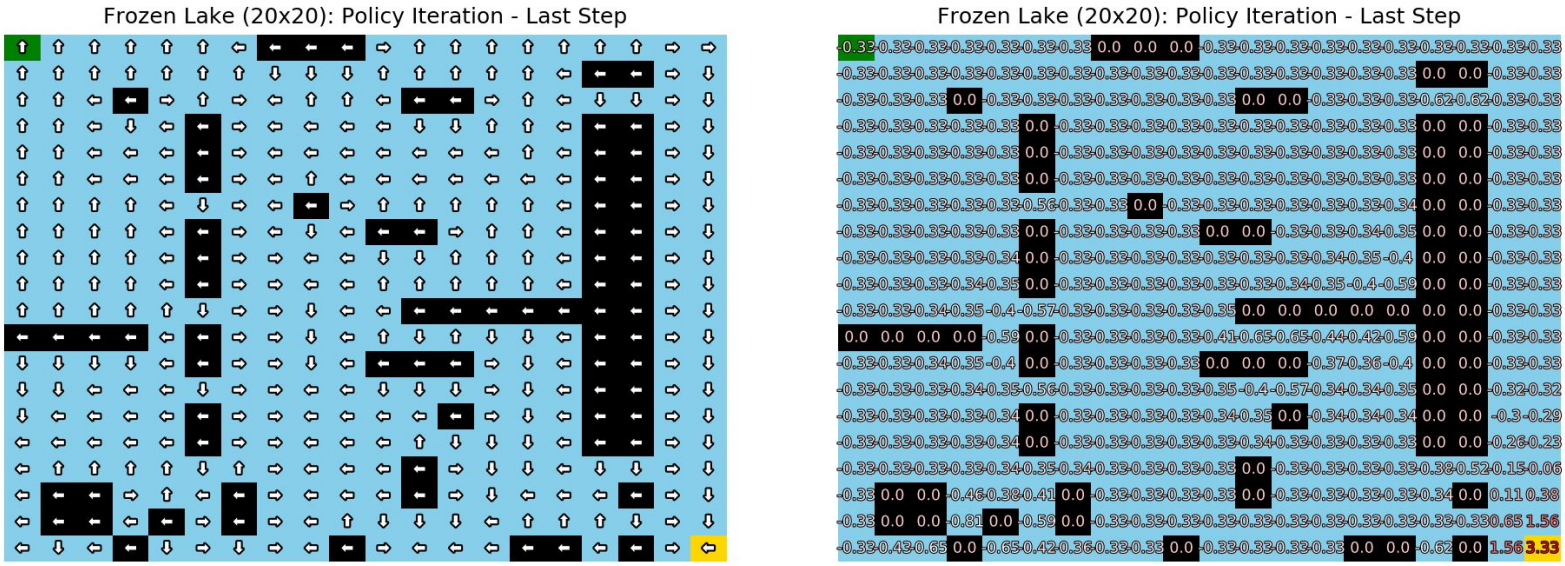


Figure 8. Final policy and values found by Policy Iteration on 20x20 Frozen Lake with a discount factor of 0.7.

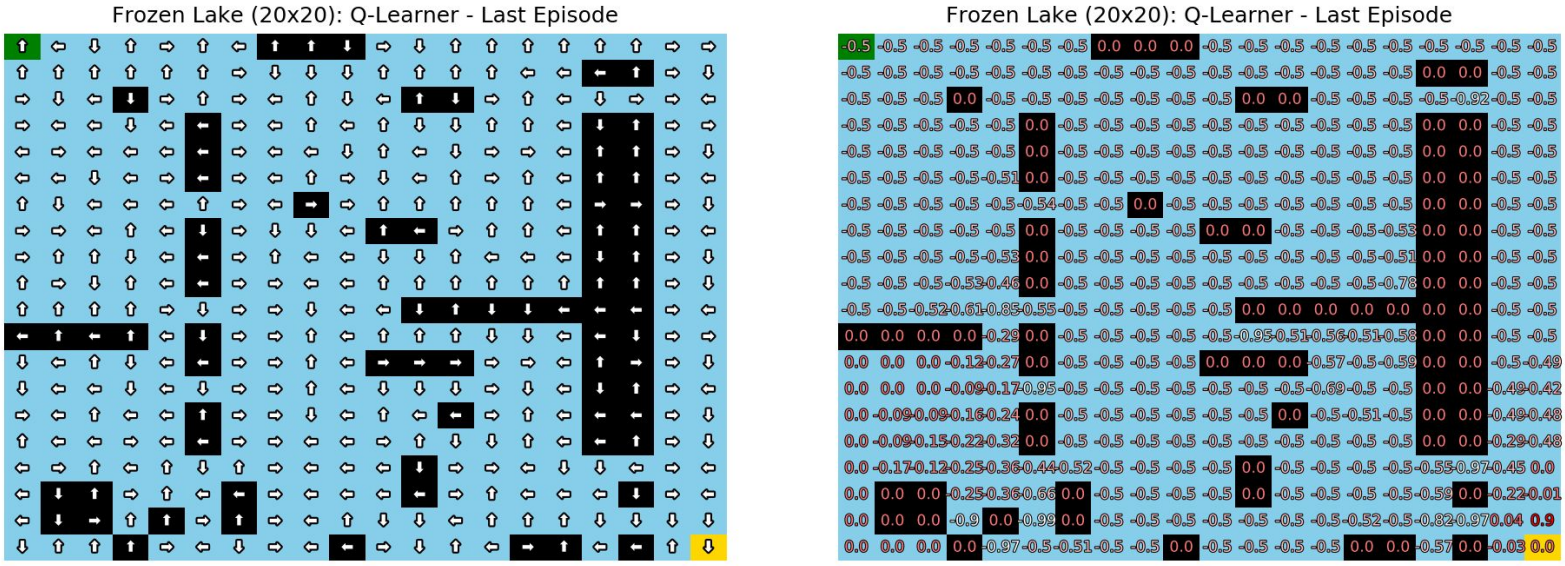


Figure 9. Final policy and values found by the Q-learning agent on 20x20 Frozen Lake with an alpha of 0.9, random initial Q values, an epsilon of 0.1, and a discount of 0.8.

The final policies learned by each algorithm are largely the same, so I would mostly like to call attention to the changes in parameters which resulted in the optimal policy with respect to mean reward. Value Iteration's discount factor increased somewhat significantly from the 8x8 version, from 0.5 to 0.7. This

makes sense, as the larger map and thus larger state space would lend itself to preferring long-term rewards. However, the end policy is similarly conservative. Interestingly enough, the discount factor decreased slightly for Policy Iteration for the 20x20 version, from 0.8 to 0.7. Since Policy Iteration and Value Iteration are similar algorithms and should, in theory, converge to the same answer, the general fact that they both found optimal results with the same discount factor is not surprising to me. I was quite surprised that the optimal discount factor decreased from 0.8 to 0.7 when moving from 8x8 to 20x20 for Policy Iteration. However, this isn't a very large deviation, and won't result in a significantly more "short sighted" agent than a larger decrease would result in. The Q-learner changed significantly. This Q-learner used a very high alpha, 0.9. This puts very little weight into previously learned values of a state, and thus makes an agent more likely to explore, rather than exploit what it has already learned. For a large state space such as the 20x20 Frozen Lake, this could be a good strategy. Rather than trust that we've already found the correct value for a state in a large state space, we should try to explore more of the state space and find something more optimal. There was also a dramatic increase in discount factor of the Q-learner, from 0.1 to 0.8. This can again be attributed to the change in size of the state space. Since the state space is large, we want to think about long-term goal, since the goal state (and hence the state with the most value in and around it) will be further away from the start than in the smaller version. An interesting change is the decrease in epsilon value from 0.5 to 0.1. This decrease makes it much likely that the agent will randomly choose an action other than the action which will bring it to the state with the highest Q value. I found this interesting because it seemingly counteracts, at least slightly, the high alpha value in that it will result in a less exploratory agent than an agent with a higher epsilon value.

We can see in Figure 11 that once again the algorithms all performed similarly with respect to average reward gained following the optimal discovered policy, with Q-learning performing very slightly worse than the others. We see more dramatic differences when looking at performance as it relates to how quickly each algorithm converged. Value Iteration saw a modest increase in steps taken to converge (34), while Policy Iteration failed to converge at all! The maximum number of iterations was set to 1000, and Policy Iteration had failed to converge by iteration 1000. This can only be attributed to the large increase in state space and may suggest that for some MDPs with large numbers of states, Value Iteration can perform faster than Policy Iteration. Q-learning similarly failed to converge by episode 1000, so it was terminated early. This is not surprising, as Q-learning failed to converge early in the 8x8 Frozen Lake as well. So, for this particular MDP, Value Iteration performed head and shoulders above the other two algorithms, beating the other two narrowly in mean reward and lapping the other two in time taken to converge.

Algorithm	Value Iteration	Policy Iteration	Q-learning
Mean Reward	-0.0983	-0.0991	-0.0998

Figure 10. Average reward gained by each algorithm on 20x20 Frozen Lake.

Now that we've seen how increasing the number of states can affect an otherwise identical MDP, let's examine the results of a different MDP, Cliff Walking.

Results: Cliff Walking

Listed below are the results of the experiment on the Cliff Walking MDP.

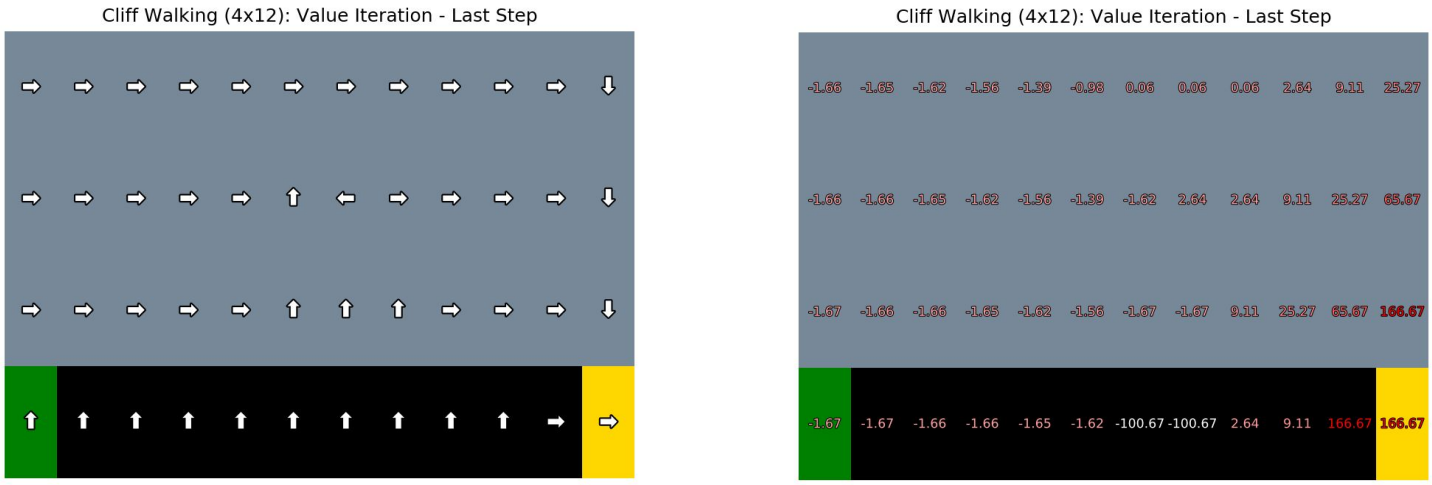


Figure 11. Final policy and values found by Value Iteration on Cliff Walking with a discount factor of 0.4.

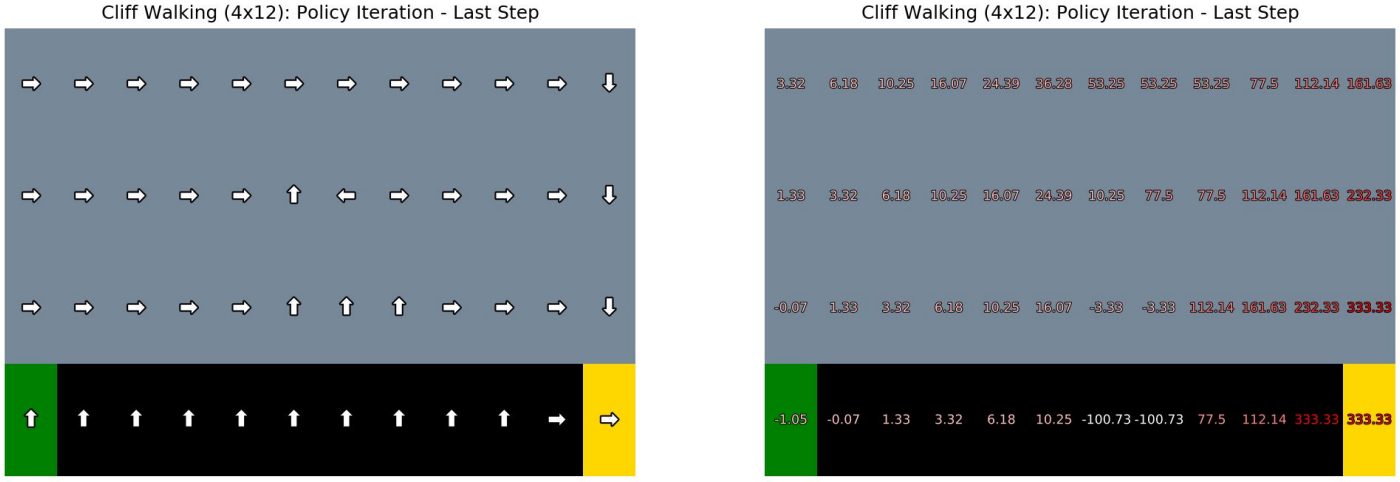


Figure 12. Final policy and values found by Policy Iteration on Cliff Walking with a discount factor of 0.7.

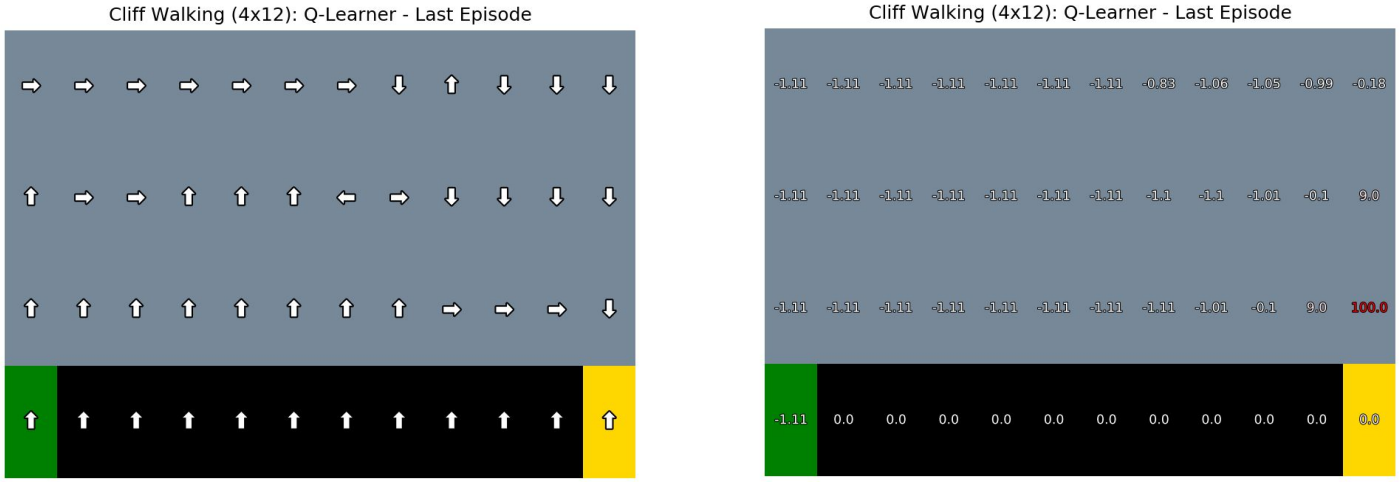


Figure 13. Final policy and values found by the Q-learning agent on Cliff Walking with an alpha of 0.5, initial q-values of 0, epsilon of 0.5, and a discount of 0.8.

In a similar phenomenon to the one observed in the 8x8 Frozen Lake experiment, Value Iteration and Policy Iteration converge to identical policies despite their large difference in discount factors. Also similarly to the 8x8 Frozen Lake, Policy Iteration converged in a fewer number of steps than Value Iteration (9 steps vs 19 steps). The Q-learner failed to converge. I think these results ended up being similar to the small Frozen Lake due to the similarity of their state spaces - the states are fully defined by the location of the agent in a grid-like environment with a single reward and certain “bad” states. The only difference is the reset mechanic when the agent goes off the cliff. Interestingly, Value Iteration and Policy Iteration don’t seem to take this into account - the policy found by these algorithms travels very close to the cliff for 5 squares. The Q-learner seems to be more mindful of this and acts more cautiously, choosing to immediately get as far away from the cliff as possible before walking in front of it. The values of the cliff states found by the Q-learner are all zero, while Policy Iteration and Value Iteration all give those states some value (some of them even got a positive value!). I think this is due to the Q-learner learning while acting within the MDP, while Value Iteration and Policy Iteration are used as planning algorithms, ran before an agent acts in the world. When the Q-learner entered a state that was part of the cliff, it was immediately sent back to the start. I am curious as to why they didn’t receive negative values, however. Given that the reward for entering a cliff state is negative, I would have guess that the value would have been negative. Perhaps if it was allowed to converge, the Q-learner would eventually arrive at a negative value for these states. The Q-learner also had a moderate alpha, 0.5, and a high discount, 0.8. This will result in a relatively exploratory agent, and is likely part of the reason Q-learning did not converge before the maximum of 1000 iterations, despite the small state space. In another parallel to the small Frozen Lake, Q-learning slightly outperformed the other algorithms with a mean reward of 5.40, followed closely by Policy Iteration at 5.376 and Value Iteration at 5.361.

Conclusion

These experiments showed that Value Iteration can outperform Policy Iteration for large state spaces in terms of convergence time and result in similar reward gained by the agent. Also displayed is the effect expanding the state space can change the optimal strategy when it comes to choosing short-term or long-term rewards. Q-learning, despite its cost in execution time, is powerful in that it requires no prior knowledge of the state space to produce similar performance with respect to reward gained by the agent.

Since this is the last assignment, I’d also like to take some space to express my gratitude towards the TAs and all others involved in running this course. You’ve all been very helpful all semester long, and I can imagine the work involved in grading the analyses of the hundreds of students in this course. For that, I just wanted to say thank you, and that I enjoyed my time in this course. Keep doing what you’re doing.