Isaac Wong
Machine Learning
Assignment 4 – Markov Decision Processes

Introduction

Reinforcement learning is a class of machine learning that is able to solve problems in the form of Markov Decision Processes (MDP). An MDP comprises of 1) a set of states, 2) a set of actions that can be taken from each state, 3) the transition model that encodes the probability of moving from one state to the next state, given a certain action, and 4) the reward model that encodes the probability of receiving a certain reward when entering a certain state. A common use for an MDP is to represent the problem of an agent moving through a world and receiving rewards during its journey. The actions that the agent will choose to take at each state can be represented by a policy. Reinforcement learning can be used to determine a policy that maximizes the total expected reward for following the policy.

Problems

Two problems were chosen: cliff-walking(CF), and frozen lake(FL). CW involves an agent starting on the bottom left of a rectangular 4x12 grid. Along the bottom of the grid is a cliff, and on the bottom right is the goal. The agent moves stochastically through the grid: each action has a 10% chance of being pushed toward the bottom of the grid. There is a large positive reward of 100 at the goal state, and a large negative reward of -100 at the cliff states. All other states have a small negative reward of -1. In addition, when the agent enters a cliff state, it is returned to the start state. This problem is interesting because it's a simplified version of what an exploratory robot might encounter in unknown terrain. It has a target where it would like to reach, and there are severe hazards along the way. In addition, there is a chance while it takes an action that the wind blows and it is blown in the direction of the wind (toward the hazard). The optimum policy that we would like to determine is one that reaches the goal as quickly as possible while avoiding the cliff.

FL involves an agent starting from the top left of a square 20x20 grid, and making its way to the goal state at the bottom right. The lake is frozen, and so the agent moves stochastically through the grid: there is a 1/3 chance of performing the action that it executes, 1/3 chance of moving to the left of that action, and 1/3 chance of moving to the right of that action. There are also holes in the ice peppered throughout the lake, where the agent would receive a moderate negative reward of -1. The goal state has a moderate positive reward of +1, and every other state has a small negative reward of -0.1. This problem is also interesting because the number of states is much larger than the CW problem, and the locations of the negative reward is more complex. The negative rewards are peppered throughout the grid, in a maze-like structure, whereas the CW problem has the negative reward lined up on one side of the grid. In both cases, the stochasticity of the transitions makes it possible for the agent to fall into the negative reward states as it takes actions, which encourages the agent to take a path that is not so close to a negative reward state.

Methods

We evaluated the 2 problems using 3 techniques to solve MDPs: value-iteration(VI), policy-iteration(PI), and Q-learning. Both VI and PI takes advantage of knowing the transition model, but in most cases in the real world the transition model is unknown or difficult to determine. Q-learning is a technique that does not require knowledge of the transition model, and learns the transition model implicitly as it takes samples from the environment. The problems and techniques were implemented in Python [1].

Value Iteration

VI is a technique that computes the long term expected reward(utility) for being in each state. It does so through multiple iterations, and in each iteration looks over all the possible actions it can take from the current state. The rewards from each state will look like it propagates outward into neighbouring states, and the utilities are eventually guaranteed to converge. The equation for value iteration is:

$$U_{t+1}(s) = R(s) + \gamma \max_{a} \sum_{s'} T(s, a, s') U_t(s')$$

Gamma is the discount factor, which adjusts how much is learned from the neighbouring states. If gamma = 0, then the utility of the state is simply the reward gotten from the state, and there is no way to include the long term expected reward into the utility. With a gamma close to 1, the policy tends to attribute more weight to long term rewards. We ran value iteration for both problems with varying discount factors. In figure 1 is the result of value iteration with a discount factor of 0. We can see that the cells for each problem are mostly identical, with no gradient in value since the rewards for the basic cells (non-cliff, non-holes, and non-goals) are the same. The induced policy will have randomly pointing arrows, and it only takes 1iteration to reach convergence.

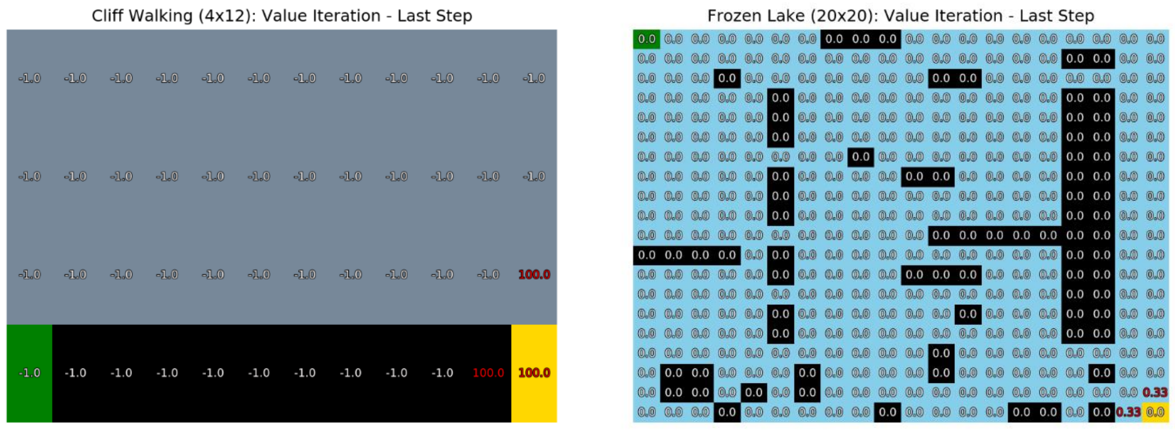


Figure 1: Value iteration for discount factor = 0

In contrast, we have the result of value iteration with a discount factor of 0.9 in figure 2. We can see from the result for CW that the goal value has propagated outward through the neighbouring cells. We also have a similar result for the FL problem, but the values further from the goal state are too small to be represented with 2 decimal places. From the results in figure 1 and 2, we can see that a higher discount factor allows the computed utility to weight the future states more heavily.

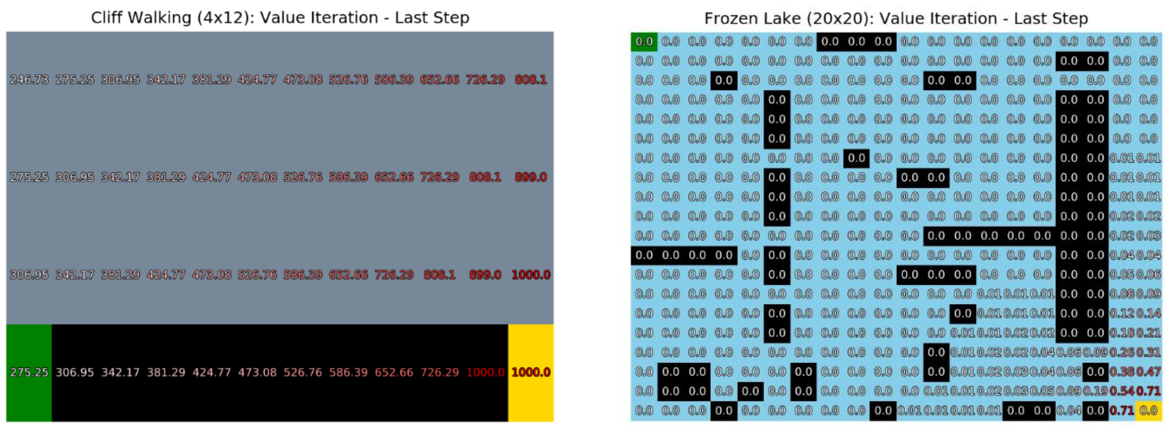


Figure 2: Value iteration for discount factor = 0.9

Policy Iteration

PI is a technique that does not directly compute the utility of states over all possible actions. Instead, it computes the utility of following a particular policy, and then improves the policy based on that utility. The equation used in the policy evaluation step is very similar to the equation for VI, and uses the same discount factor. The difference is that the computation for utility does not need to look at all possible actions from a particular state, only the action that the policy says should be taken.

$$U^{\pi}(s) = R(s) + \gamma \sum_{s'} T(s, \pi(s), s')U^{\pi}(s')$$

The discount factor here plays the same role here as it does in VI. Figure 3 shows the policy determined using PI with a discount rate of 0. The arrows are initialized to point up for CW, and point left for FL. We can see that very little has changed, for the same reason that the result for VI has no differences in the values between cells.

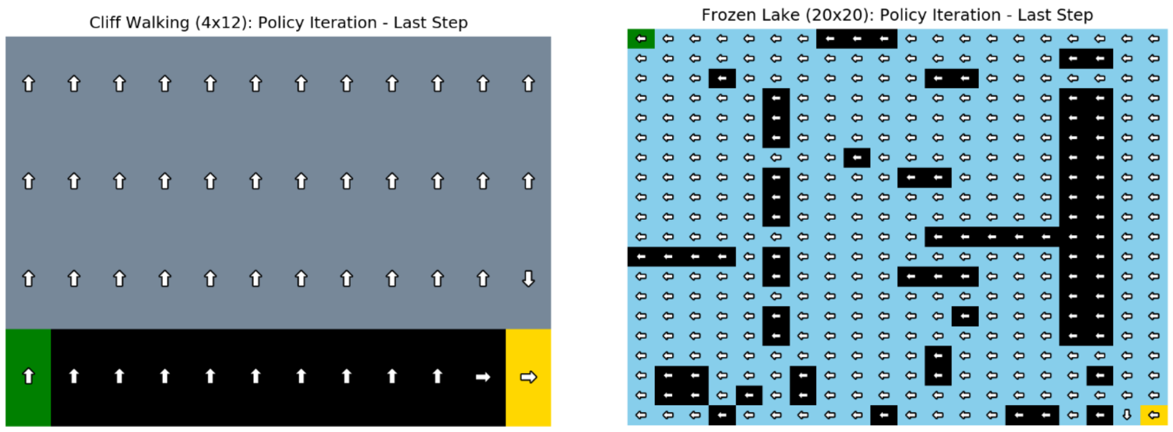
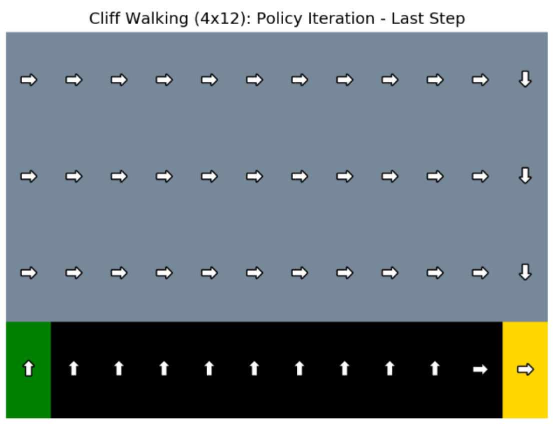


Figure 3: Policy iteration with a discount rate of 0

In figure 4 we have the results for a discount rate of 0.9. We can see that for the CW problem, the arrows are pointed to the right in most of the states, and a clear path can be seen from the start to the goal state. However, it appears that the chance of falling into the cliff is small compared to the overall reward of reaching the goal state as soon as possible. On the other hand, for the FL problem, we can see that the arrows point away from the negative reward states as much as possible. Because each action executed has an equal chance of going to the left, right, or executed direction, the block that the arrow points away from is the one that the agent wants to avoid the most.



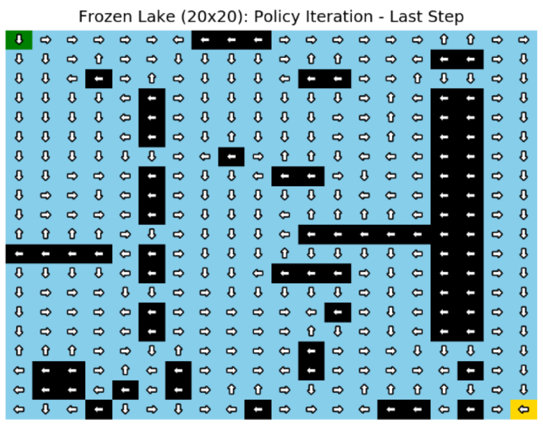


Figure 4: Policy iteration with a discount rate of 0.9

Q-learning

Q-learning is a reinforcement learning technique that does not rely on knowing the transition model. It is also an iterative algorithm just like VI, but instead of computing the utility of the current state by looking at all the actions, it chooses to take a single action. By taking this action, it takes a sample from the environment with the current estimated utility of taking that action. Over time, these samples implicitly build a transition and reward model. Unlike PI, where the action is chosen based on the policy, there is no policy to guide us since the transition model is not explicitly known. Different ways of choosing actions reflects the willingness to explore vs exploit, and will affect the outcome of the model. In this implementation, we chose to use the epsilon-greedy approach of selecting actions. Other methods include selecting an action based on the Boltzmann distribution, but this was not implemented due to time constraints. For epsilon-greedy, when selecting an action, there is an epsilon chance of selecting a random action. If a random action is not chosen, then the current best action is chosen. After each iteration, epsilon decreases. With the epsilon-greedy approach, actions are chosen randomly with a higher chance to explore the environment in the early iterations. In the later iterations, actions to exploit the current knowledge more often. The equation for updating our estimate of utility, Q, is shown below:

$$Q_{t+1}(s,a) = (1-\alpha)Q_t(s,a) + \alpha[R(s') + \gamma \max_a Q_t(s',a)]$$

Alpha is the learning rate, which is generally set to be close to 0. If alpha is 0, then there is no learning from the action taken. On the other hand, if alpha is 1, then there is no memory of what was learned in the past, and the estimate is always the reward and Q returned from the most recently chosen action. Like before gamma is the discount factor, which we have seen to be effective with a value close to 1.

The initial estimates for Q is important. In this implementation, we ran Q-learning with an initialization of 0, and a random initialization. When initializing with zero, the first choice of action will be random. If the result that returns is negative, then the next iteration will choose an action randomly that is not the last action. If the result that is returned is positive, then the next iteration will reinforce choosing that action, depending on the value of epsilon. When initializing randomly, the first choice of action will be random as well, but if that action is suboptimal, it may take a very long time for it to be reduced enough for a better option to arise. We can see from figure 5, that zero initialization works better for the CW problem than the LW problem. In the random initialization, the Q at the start state is 15.16, whereas the Q at the start state for zero initialization is 21.07. Another method of initialization is by initializing to a high Q. This has similar behavior as initializing to zero, but was not implemented due to time constraints. The Q values for the FL problem is not shown, because the values were too small to be rendered in the plot.

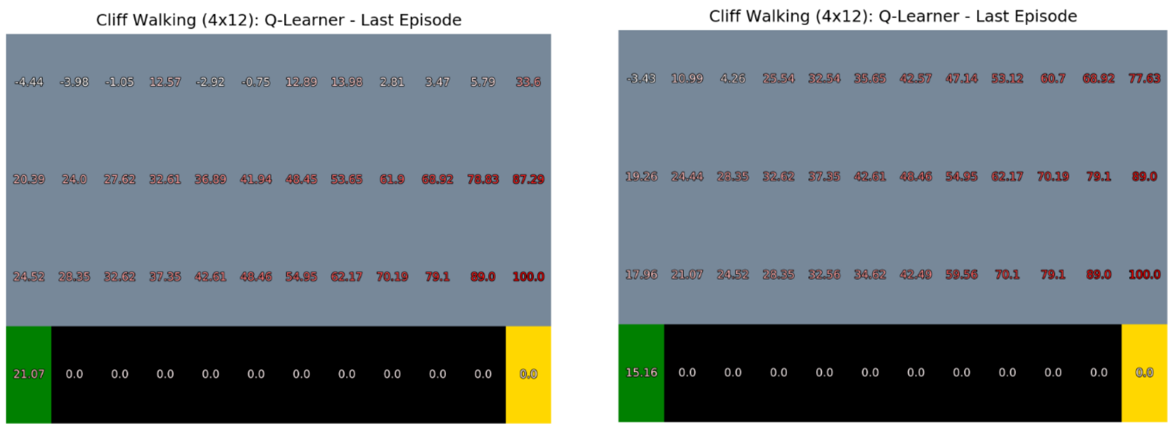


Figure 5: Q-learning with random(left) and zero(right) initialization.

We found that the value of alpha was not important in many parameter configurations for the CW problem. We varied alpha from 0.1 to 0.9 with a step size of 0.1, gamma from 0.1 to 0.9 with a step size of 0.1, and epsilon from 0.1 to 0.5 with a step size of 0.2. There wasn't a noticeable change in Q at the start state for different values in alpha, which suggests that learning more from immediate actions is comparable to remembering more from past actions. This may indicate that the actions taken tend to be more reliable, or produce less random results. Perhaps this is due to the fact that each action is mostly reliable – the action that is executed is the action that is carried out 90% of the time. For the FL problem, the values of Q were so low across the board that it was difficult to conduct analysis on the results.

The epsilon-greedy approach to selecting actions is intended to encourage exploration in the early iterations of the algorithm, and more exploitation in the later iterations. We can see this in figure 6, which shows the time taken per episode. An episode starts when the agent leaves the start state for the first time, and ends when it enters the goal state. For the CW problem, we can see that the first episode is the longest, where most exploration is done. As the number of episodes grow, less and less exploration is done, and the agent makes it quickly from the start to the end state. This is the behavior that we would like to encourage with our epsilon-greedy approach. However, for the FL problem, we see that there is no obvious trend in time per episode as the number of episodes grow. This is likely due to the problem itself – the agent only has a 1/3 chance of going in the direction that it wants to go, and a 2/3 chance of going in another direction. Even with a plan that would be the shortest path from start to goal, the agent is very likely to slide in a different direction and travel for some time before eventually making it to the goal. This shows that even with an approach that is designed to exploit in the later iterations, the transition model makes it difficult for the agent to exploit its knowledge.

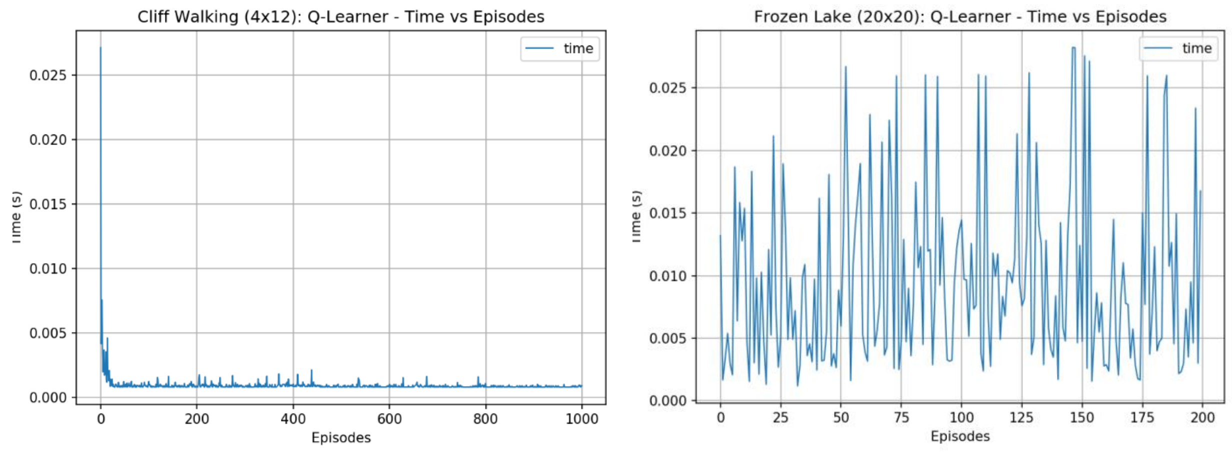


Figure 6: Time taken per episode(agent going from start to goal)

Comparison

When finding the optimal policy, the time taken to converge is also important. It's difficult to compare PI/VI against Q-learning, since Q-learning does not have knowledge of the transition model. However, we can compare PI against VI. In general, PI takes fewer iterations to converge than VI, but each iteration of PI is much more computationally intense. The time taken for PI versus VI is therefore difficult to predict.

In table 1 is the total time taken and number of iterations to convergence. The total number of iterations for PI is much less than that of VI, and for these two problems, the total time taken is longer using PI. The average time per iteration for PI is longer than that of VI, due to the policy evaluation step. It is usually done by solving a set of linear equations, which can be an operation that is cubic with the number of states. Unsurprisingly, we also see that the time per iteration for both VI and PI increases with the number of states.

Table 1: Total time taken and iterations for PI and VI for the two problems			
	Total Time Taken(s)	*Average Time per Iteration (ms)*	*Total Number of Iterations*
PI (Cliff-Walking)	0.43	61	7
VI (Cliff-Walking)	0.10	0. 65	154
PI (Frozen Lake)	1.69	240	7
VI (Frozen Lake)	0.36	6.8	53

The policies determined through VI, PI, and Q-learning for the CW problem are shown in figure 7. We find that the policies for PI and VI are identical, and the policy for Q-learning remarkably similar to the two. Along the cliff, all policies have determined that it walking along the cliff has the best long term expected reward despite the chance of being blown into the cliff. The similarity of the policies for PI and VI is not surprising, since they are both convex optimization problems that should give the same optimal result. It is not so clear that Q-learning is a convex optimization problem because of the stochasticity of selecting actions, but the resulting policy is not much different from that of PI and VI.

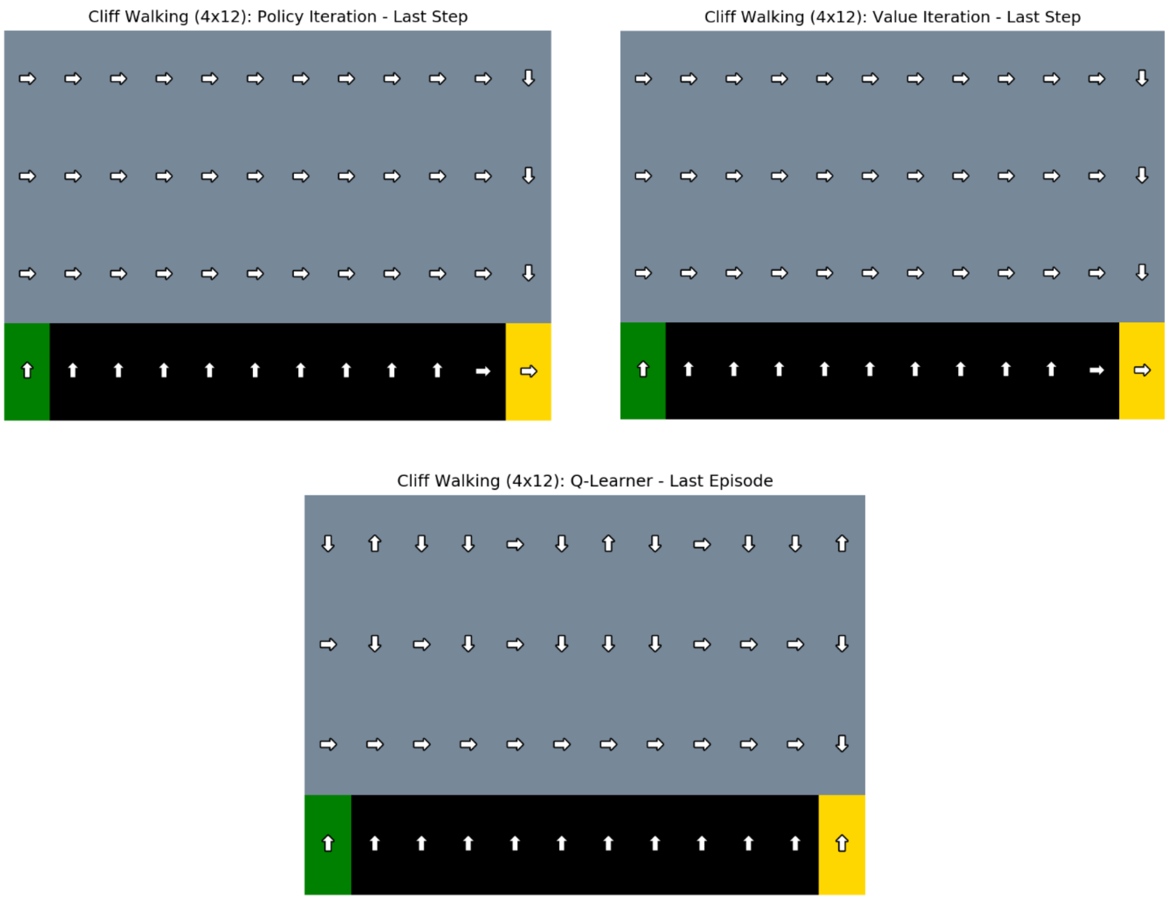

Cliff Walking (4x12): Policy Iteration - Last Step
Cliff Walking (4x12): Value Iteration - Last Step
Cliff Walking (4x12): Q-Learner - Last Episode

Figure 7: Policies determined through PI (top, left), VI (top, right), and Q-learning (bottom).

The policies determined through VI, PI, and Q-learning for the FL problem are shown in figure 8. Just like before, the policies for VI and PI are identical. However, Q-learning did not do well in determining a policy. For this problem, random initialization appeared to work better. However, m any of the arrows still seem to be pointing randomly. In the map for the converged Q-learning values, we can see that all the values are very close to zero, but we can see the relative magnitudes by looking at the gradient of colour. The darker red values are larger in magnitude than the lighter red values. There is a higher concentration of dark red near the goal state, and a higher concentration of light red near the start state, leading to an overall flow from the top left to the bottom right. There are also many cases where states point toward negative reward states. It is difficult to determine why Q-learning doesn't do well for this, but it may be possible that the small magnitude of the rewards and the large number of states makes it difficult for the goal reward to propagate back.

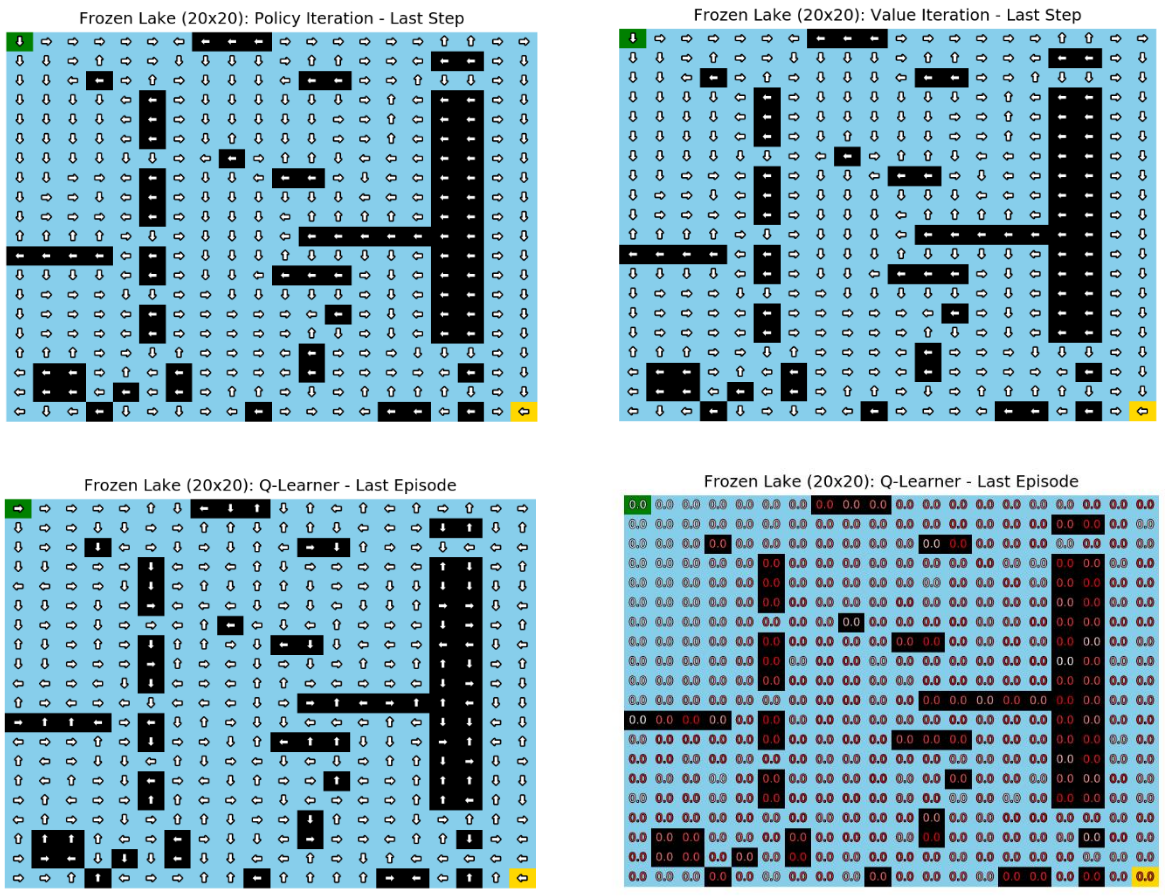

Frozen Lake (20x20): Policy Iteration - Last Step
Frozen Lake (20x20): Value Iteration - Last Step
Frozen Lake (20x20): Q-Learner - Last Episode
Frozen Lake (20x20): Q-Learner - Last Episode

Figure 8: Policies determined through PI (top, left), VI (top, right), and Q-learning (bottom, left). Q-values (bottom, right)

Conclusion

PI and VI are both ways to find the optimal policy, through either computing the utilities of the states explicitly, or iteratively improving an arbitrarily initialized policy. Both are convex optimization problems, and should be expected to arrive at the same result. This was shown in the comparison section above. The choice to use PI or VI should therefore be how long it takes for the algorithm to converge. PI uses fewer iterations, but has a high computational cost per iteration (cubic in the number of states). VI on the other hand has a much larger number of iterations, but a low computational cost. For the problems that we studied, VI had a lower convergence time overall.

Q-learning did not perform as well as PI and VI for determining an effective policy. This can easily be attributed to the fact that a Q-learning model does not have explicit knowledge of the transition and reward models, while PI and VI models do. However, Q-learning worked well for the small state space in the CW problem, forming a policy that was remarkably similar to that of VI and PI. In a larger state space, and possibly due to the values of the rewards, Q-learning did not do well for the FL problem. This shows the susceptibility of Q-learning to not do well in certain conditions, and that parameter tuning of alpha, gamma, and the initialization has to be done carefully in order to get good results.

Sources

[1] Source code from: https://github.com/cmaron/CS-7641-assignments