CS7641: Analysis and Report for Assignment 4- Markov Decision Processes

Mingming Zhang mzhang607@gatech.edu

Introduction

This assignment is to understand how the value iteration and policy iteration are operating during Reinforcement Learning applications. On top that, we will also apply Q-learning to compare the differences when using policy iteration and value iteration. In this assignment, I will use two different approaches to solve two different problems.

The first problem I choose is the *Frozen Lake* problem. For this problem, I will use the module OpenAI's Gym to generate the problem itself. The document for the module can be found here. https://gym.openai.com/. The reason I think it is interesting because it is a discrete finite Markov Decision Process problem (MDP) representing a classical grid world problem, with straightforward intuition to explain solving an MDP. It can help me to understand the basics of Reinforcement Learning. To solve this problem, I will implement value iteration, policy iteration and Q-learning by coding from the definition.

The second problem I choose is the *Forest Management* problem. For this problem itself and solving this problem, I will use the packages from the *Markov Decision Process (MDP) Toolbox*. https://pymdptoolbox.readthedocs.io/en/latest/api/mdptoolbox.html. The reason I am interested in this process is that it is a non-grid world, sequential MDP problem. In addition, the forest management problem also reflects a real-world environmental application, where I think it might have some practical value.

In addition, I also want to compare my implementation of policy iteration, value iteration and Q-learning to those of the *MDP Toolbox*, to see if I observe similar computation expense behavior.

The *Frozen Lake* problem

Overview of the "Frozen Lake" problem

The *Frozen Lake* problem is an 8 by 8 grid world problem. The agent controls the movement of a character in a [8x8] grid world, a state space with 64 discrete cells (Fig.1). The agent needs to move from starting cell "S" to ending cell "G" (goal). Actions the agent can take are four discrete movements, left (0), Down (1), Top (2) and Right (3). A cell "F" (frozen surface) means it is walkable and the agent will get a reward of (1) if walks on it. A cell "H" (hole) means the agent will fall into it and the game is over. The movement direction of the agent is uncertain, and it is partially dependent on the chosen direction.

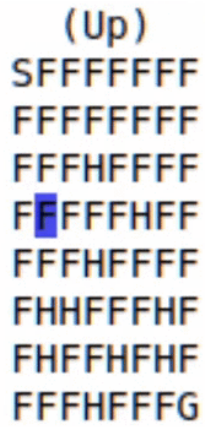

Figure.1 Illustration of a "Frozen Lake" problem, a [8x8] grid world problem. "S" stands for starting state. "G" stands for ending state. "F" stands a cell in the grid is walkable. "H" stands for a cell is with water and agent will fall into it. The figure here shows the agent is currently in the cell highlighted in blue and the next action it is going to take is mobbing up.

At each state in this problem, there is a corresponding transition probability to end up in its successor state. Let's take a look at some examples from the transition probability matrix. In Table 1 left column, it shows when the agent is state (1), taking action (1), it will have equal chance of 33.3% ending in state (0), (8), (1), respectively. After taking each action, the reward is 0, and the action is not done, because these three states they are neither "H" (hole) state or "G" (goal) state. Let's take a look at the example in the right column in Table 1. From state (12) taking action (3), there are two scenario the game could be over. Reaching state (42) and (49), both are "H" (hole) state. The agent will fall into the water and thus, game is over. Of course, in these two states, the immediate reward is zero for both cases.

Table 1: visualization of some transition probability in the *Frozen Lake* problem	
env.env.P[0][1] In state (0), take action (1)	env.env.P[12][3] In state (12), take action (3)
(0.3333333333333333, 0, 0.0, False)	(0.3333333333333333, 51, 0.0, False)
(0.3333333333333333, 8, 0.0, False)	(0.3333333333333333, 42, 0.0, True)
(0.3333333333333333, 1, 0.0, False)	(0.3333333333333333, 49, 0.0, True)

Overview of value iterations

In value iteration, the goal is to find the optimal value function, by starting off with some random value functions, like all zeros. Then we iterate the process using Bellman Optimal Equation, computing the new value function of a state by plugging the values of previous iteration in the Bellman Optimal Equation. This process is repeated until the change in the Optimal Value function is very small, let's say than an arbitrary small number θ. Then we call the Value function has converged. The converged value function is the optimal value function we are trying to find.

During each iteration of calculation for finding optimal value function, this process will not compute the policy for every intermediate value function. Once we find the optimal value function, we perform just one sweep over all the states and act greedily with current value function to find the optimal policy.

Overview of policy iterations

In Policy Iteration, similar to the process that is in Value Iteration, it will try to find a better value function during each iteration of calculation. The major difference is that after each iteration, the process will evaluate the policy in each iteration.

When it comes to update the value function during each iteration, instead of using the expected values resulted from all possible actions in Value Iteration, Policy Iteration will use the current policy to generate an action, and use the generated action resulted value to update the value function. The it uses the updated value function to improve the policy through all the states by acting greedily. This process will stop when the policy from the new iteration is the same as the previous iteration. Then the policy we find is considered as the optimal policy.

Thus, we can treat that Value iteration can be seen as a special case for Policy Iteration, where the process of policy evaluation is stopped after one step.

Overview of Q-Learning

Q-learning is different than both Value Iteration and Policy Iteration when solving a MDP problem. In Q-learning, because it assumes the transition probability from one state to the successor states are not available, it constructs a Q-table where it hosts all the state-action-rewards value pairs it has observed. Because there is no transition probability matrix that can be utilized, Q-learning is purely depending on the process of the agent interacting with the environment and update its knowledge from its experience with the environment.

Q-learning starts with a table full of zeros (size of (number of states, number of actions)). Each value in the table represents the q value of a given state and taking a certain action. As the agent explore the environment, it will update the table value after each iteration. Because it is experience based learning, as a consequence, Q-learning usually needs a large number of iterations to accumulate the knowledge from the environment when solving a problem.

Experimental results using value iteration, policy iteration and Q-learning in solving the “Frozen Lake” problem

Based on the definition of Value Iteration, Policy Iteration and Q-learning, I implement the code from scratching when solving the *Frozen Lake* problem. The parameter used for Value Iteration and policy iteration is as following, gamma=0.99, theta=0.0001. Figure 2 shows the results of the optimal policy generated by using Policy Iteration and Value Iteration. From the results we can

see that, the optimal policy generate by both of mine implemented methods are very similar. There are some minor differences that has been highlighted in the Cyan squares. The results make sense to me as the definition describes these two methods shares a lot of common iteration process. The minor difference that I found in these two optimal policies could be due to the differences in Policy Iteration where each value function update, there is a policy evaluation happening that is updating the existing policy, while there is no such policy updating process in value iteration until the value function has converged. This could explain why there are several states the action is taken differently when using value iteration and policy iteration.

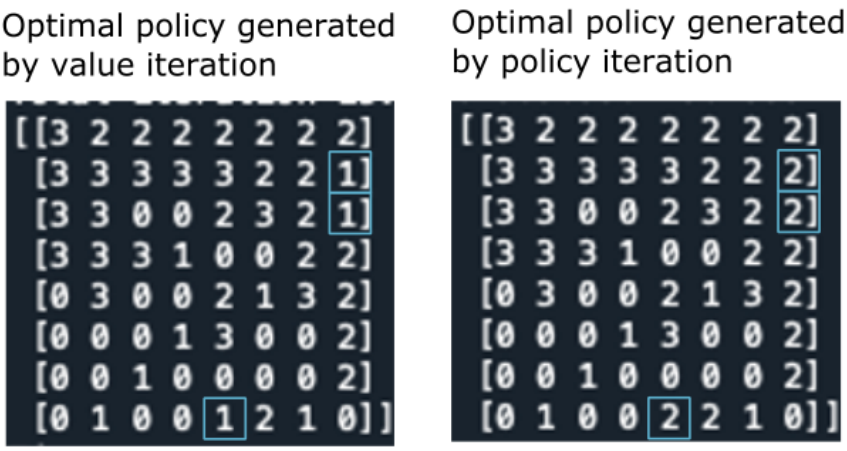

Figure.2 Optimal policies generated by using Value Iteration and Policy Iteration approaches. The environment is an 8x8 grid word. Each value in the matrix represents the action the agent should take within each optimal policy. Blue square marks the differences between these two optimal policies.

Next, I ran experiments with 500 episodes, using the optimal policies I found to solve the *Frozen Lake* problem. Table 2 shows the results from all the experiments. We can see that out of 500 episodes, when using optimal policy generated by Value Iteration, we can reach the "G" (goal) state successfully 421 times out of 500 episodes. When using Policy Iteration generated optimal policy, the agent reaches the "G" (goal) state 416 times out of 500 episodes. The successful rate of the agent reaching the "G" (goal) state without falling into the "H" (hole) is very similar, 84.2% V.S. 83.2%.

When we look at the number of iterations that is needed to converge to the optimal policy, we see that using Value Iteration method it will take 23 iterations to converge, while it will take 9 iterations to converge when using Policy Iteration based method. Because the number of iterations is much fewer when using Policy Iteration compared to when using Value Iteration based method, Policy Iteration found optimal policy also converge much faster in time, taking 0.0045 s in my experiment (Table 2). I also tested the number of iterations Policy Iteration and Value Iteration will take when using different discount factors. The experiment indicates Policy Iterations will use fewer number of iterations than Value Iteration in finding optimal policy (Fig.3). I think the reason could be that the act greedily action in Policy Iteration is strategically moving

toward the direction, where it can have the largest state-action value, which helps the policy to converge in a much faster way. This acting greedily action is not available in Value Iteration based method, where it purely depends on the expectation of the values over all actions, which is less strategically and will spend more time to converge.

Table 2: experimental results in solving *Frozen Lake* problem using different methods				
	Time cost to find the solution (s)	**Total number of iterations until find a solution**	**Succeed in reaching the goal state over 500 episodes**	**Succeed rate over 500 episodes**
Value iteration	0.234	23	421	84.2%
Policy iteration	0.045	9	416	83.2%
Q-Learning	87.277	4904137	319	63.8%
Q-Learning (2)	455.3006	26893208	233	46.6%

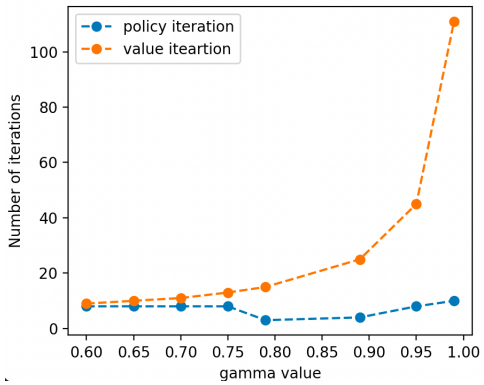

Figure.3 Number of iterations when solving optimal policy using different discount factor (gamma) values. The experiment indicates policy iteration is using fewer number of iterations to find the optimal policy across all the gamma values I tested.

Next, I ran the experiment with Q-learning that I implemented from scratch. The alpha parameter in Q learning was set at 0.628. Here each iteration means the Q value table has been updated once. We can see that in order to find a Q value table from exploring the environment that is closer to the performance compared to Value Iteration and Policy Iteration, it took 4904137 number of iterations to get successful trials of 319 out of 500 episodes (Table 2). The successful rate at this point is 63.8%, which is way less than 84.2% and 83.2% when using Value Iteration and Policy Iteration. In a different experiment (Table 5, **Q-Learning (2)**), where I increased the number of iterations that I used to update the Q-tables. The performance is not improved, as I only observe 233 successful trials out of 500 episodes. I think the reason is because Q-table is updating based on what the specific experience is in each step of exploring the environment, agent is not always in the right path to the right solution, where simply increase the number of iterations might not help the agent significantly.

However, I would say in a lot of real-world problems where the transition probability matrix from one state to the successor states is often not available. Thus, Q-learning still have some practical applications as it just needs to explore the environment.

The *Forest* problem

Overview of the "Forest" problem.

The forest is a problem managed by two actions, 'wait' and 'cut'. An action is decided at each state (a certain time period) whether to maintain an old forest for wildlife or cut the forest to sell for money. However, at each age stage, there is a probability that a wildfire might happen, and all the woods might be burned.

For all the experiments, the future reward discount rate is 0.8, and the total number of age states are set to be 800. The probability that a wildfire will happen in the near future is 0.05. In this problem, there are two types of reward that one can experiment with. The first one is for the 'wait' action at the state when the forest is the oldest, the action is wait (r1). The reward is encouraging the action to preserve the forest for wildlife. Another reward is for the 'cut' action at the state when the forest is the oldest. This reward is encouraging cutting the woods and sell it for the short-term economic benefits.

Experimental results when encouraging preserve the forest.

First, we will set the 'r1' reward to be 50, and 'r2' reward to be 2. This strategy encourages the manager to keep the forest as it will give a much higher reward if the action at the oldest is wait, than the 'cut' action at the forest's oldest state.

Table 3: experimental results in solving *Forest* problem using different methods when encouraging preserve the forest for wildlife

(Parameters: fire_rate = 0.05, discount=0.8, 800 states, r1=50, r2=2)

	Time cost to find the solution (s)	**Total number of iterations until find a solution**	**Expected reward at the last state**
Value Iteration	0.0136	37	208.685
Policy Iteration	0.2246	17	208.693
Q-Learning	1525.948	30000000	143.767

Table 3 shows the summary of this experiment. From a very high level, we noticed that Value Iteration and Policy Iteration based methods from ***mdp ToolBox*** resulted in a very similar long term expected reward at the oldest state. When using Value Iteration, the expected reward at the oldest state is 208.685, while that from using Policy Iteration is 208.695. The overall expected reward over the number of states is very similar in the results using these two methods as well

(Fig,4 a, b), which is similar compared to that results when solving the *Frozen Lake* problem shown above. The two optimal policies found by using value iteration and policy iterations are very similar, with some action to preserve and cut the forest at the early and oldest stages, while taking actions to cut the forest in between. I think a big reason is that there is 5% chance to have wildfire at each stage, and the expected reward will be higher if cut the forest, rather than get it burned by wildfire.

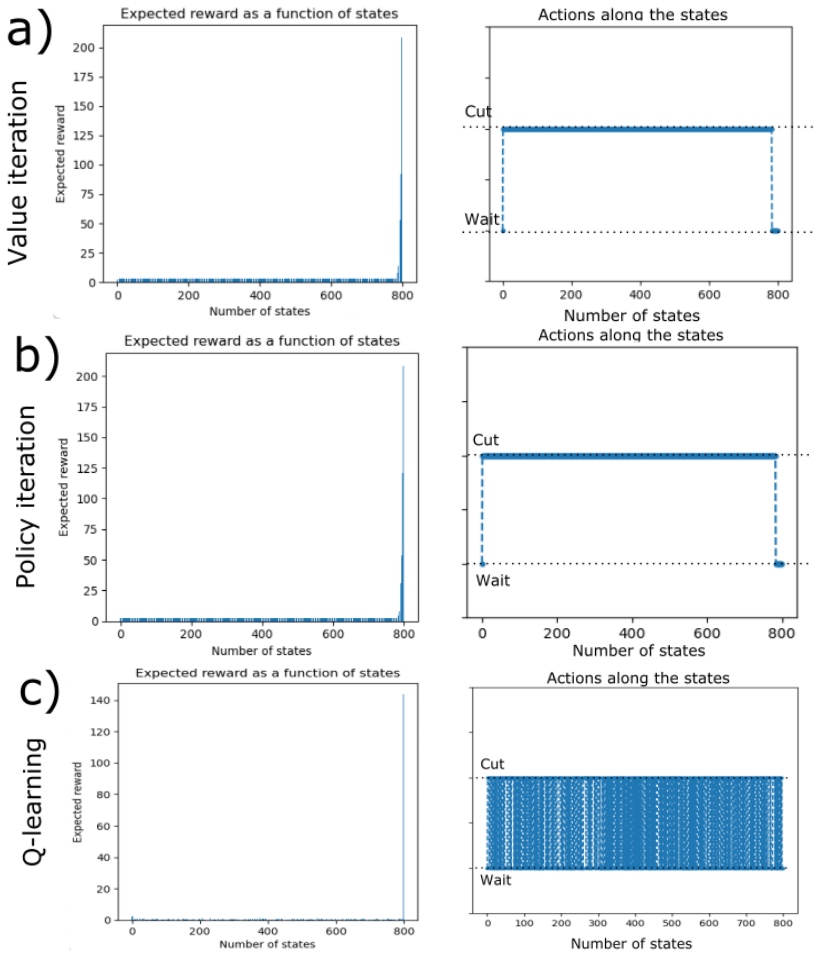

Figure.4 Experimental results focus more on preserve wildlife rather than short term economic benefits. Here the parameter I used are as follows. There is 5% chance in the near future could have a wildfire in the woods. When the forest reaching the oldest state and the action is 'wait', the reward is 50; while the reward is 2 when the action is cut at that oldest 'state'. This setting is meant to encourage preserve the woods for wildlife. Each row shows the results using a certain method to solve the problem. Left column in each row shows the expected reward after each state of oldness. Right column in each row shows the planned action that is executed along the states of the oldness.

In terms of time consumption, policy Iteration will take fewer number of iterations compared to Value Iteration to find the optimal policy. If we look at the time consumption when using these two methods, Policy Iteration will take overall longer time (0.2246s) to find the optimal policy compared to using Value Iteration (0.0136s) (Table 3). If we look at the results from solving *Frozen*

Lake problem in Table 2, we see that using Value Iteration will take longer time to find the optimal policy, which is opposite from what I observe here in Table 3. I think one of the reasons is that because I implement every from scratch when solving *Froze Lake* problem, the code was not optimized with a lot of for-loops. There are a lot of for-loops in my implementation, which are straightforward to understand, but not the best coding practice that can be implemented. Thus, it will take longer time if there is any matrix manipulation that can be done. My understanding is, within each iteration, the policy iteration will take longer time because it will act greedily to choose the current action and then evaluate the current policy to see if it converges (policy becomes stable) before moving on to next iteration. Thus, policy iteration will take much longer time in each iteration. And it might take longer time when the number of iterations is fewer than that will take in the value iteration.

Table 3 also shows Q-learning takes much longer time to find a policy, while the performance is no near close to the that from using Value Iteration and Policy Iteration (also see Fig.4c). Similar to what we observe when solving *Frozen Lake* problem, the experience-based Q-learning will need a significant higher number of iterations to find a policy. If we can want to improve the performance to get a better policy, the cost of time to improve is very high. The actions Q-learning is taking along the state of oldness is much more chaotic and we can see the action is alternating frequently between 'cut' and 'wait' (Fig.4c). This is expected, as the Q-table is updating based on the what the agent is exploring at a particular time.

Experimental results when encouraging cutting the forest for short term benefits.

Next, I change the parameter to make an assumption that we want to cut the forest to make short-term benefit, instead of preserving it for wildlife (keep r1=5, r2=50). I observed similar results compared to what was shown above. Policy Iteration and Value Iteration can find similar policy which can result in similar award at the very oldest state (Table 4, Fig 5, a and b). The actions these two optimal policies will take are very similar or the same along the state of oldness of the forest (Fig 5 a and b). Policy Iteration will find the optimal policy in fewer number of iterations, while value iteration will take the least amount of time to find the optimal policy (Table 4).

Table 4: experimental results in solving *Forest* problem using different methods when encouraging cut the forest for short-term benefit

(Parameters: fire_rate = 0.05, discount=0.8, 800 states, r1=5, r2=50)

	Time cost to find the solution (s)	**Total number of iterations until find a solution**	**Expected reward at the last state**
Value Iteration	0.00846	23	51.713
Policy Iteration	0.1621	12	51.727
Q-Learning	1529.751	30000000	12.06

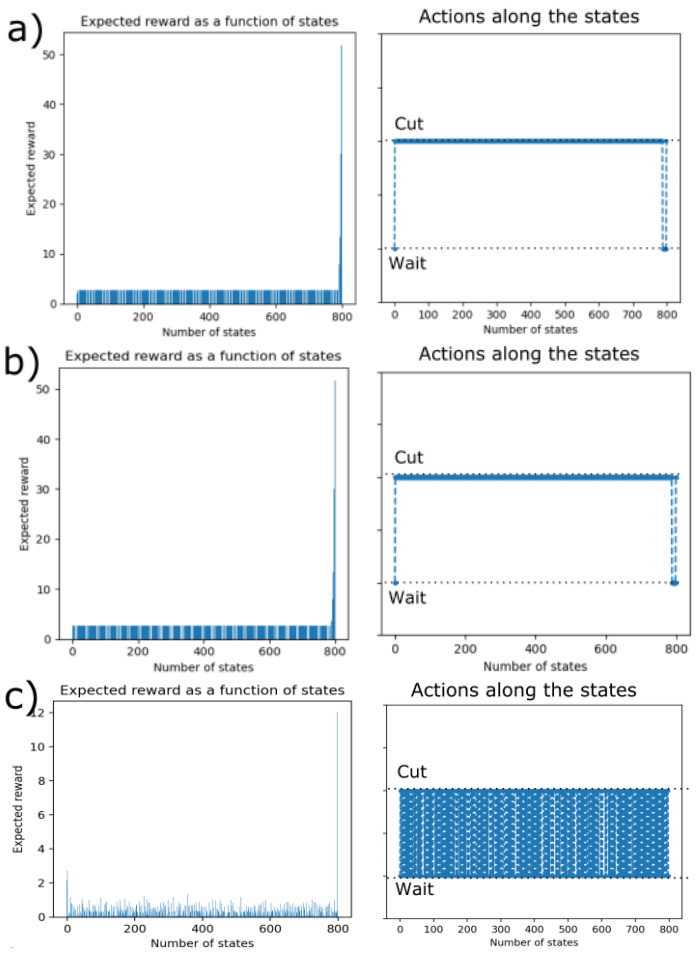

Figure.5 Experimental results focus more on cutting the woods for short term economic benefits. Here the parameter I used are as follows. There is 5% chance in the near future could have a wildfire in the woods. When the forest reaching the oldest state and the action is 'wait', the reward is 5; while the reward is 50 when the action is cut at that oldest 'state'. This setting is meant to encourage preserve the woods for wildlife. Each row shows the results using a certain method to solve the problem. Left column in each row shows the expected reward after each state of oldness. Right column in each row shows the planned action that is executed along the states of the oldness.

In contrast, Q-learning will take number longer time to find a policy that have reward at the same number of magnitudes, but the reward is nowhere close to the rewards that we can get from Policy Iteration and Value Iteration Table 4 and Fig.5c.

The policy resulted action along the states are similar when using Value Iteration and Policy Iteration, while the sequence of action is very different in the results get from using Q-learning.

Conclusion

Based on above experiments and results, we can see that Value Iteration and Policy Iteration uses a very similar fashion of iterations to update the value function. In Policy Iteration, the policy will be evaluated and updated after each value function has been updated. This is the major difference between Policy Iteration and Value Iteration. Thus, Policy Iteration tends to spend fewer number of iterations to find the optimal policy, but with a cost of higher computational time in each iteration, as they have to act greedily to update the policy. Overall, Policy Iteration and Value Iteration could converge to very similar or the same optimal policy. The performance of these two methods is usually very similar.

On the other hand, Q-learning will take much longer time to find the solution. In Q-learning, the transition probability matrix from one state to the successor states is not available and Q-learning is relying on exploring the environment to update its knowledge (Q-table). Thus, there is no clear strategy to converge its solution rather than using high number of times in exploration. As a results, the number of iteration Q-learning needs are significantly higher compared to Policy Iteration and Value Iteration. The performance of Q-learning in my experiments above are not nearly as good as using other two methods. However, one advantage in the real-world practice is that Q-learning does not require the knowing of transition probability, which oftentimes it unknown in real world problem. From that perspective, Q-learning might be more practical.