Name: Sol Won

Machine Learning – Markov Decision Process

Description of the problems:

Reinforcemet learning is principally expressed by probability model called Markov Decision Process (MDP). MDP uses probabilities and graphs to model decision processes. Also, it is designed based on first-order Markov assumption (state of time t is only influenced by state of t - 1). MDP can be defined by <S, A, P, R, γ> tuple (state, action, probability, reward and discount factor).

Figure 1: MDP Flow Diagram

Artifical intelligence, robots, systems are considered as agent and agents perform A_t action of S_t state. Then, environment (game rules, engines) return S_{t+1} state and R_{t+1} reward to an agent. Goal is to choose an action with optimal total rewards in each state and find optimal policy π to express the distribution of actions in the states. Important thing in MDP is that it knows the model (observation). Model here can be defined as transition probability and reward function.

For this assignment, two interesting MDPs are solved by three different algorithms. First, forest management example from mdptoolbox is used. There are two actions that can happen in this forest and those actions are decided each year. Wait action is to save wildlife and cut is to make money. There is one more interesting parameter which is the probability of fire burn in the forest. I found this problem interesting since the problem has small number of states and also, it is non-grid world problem. As a default, the state is 3x3 but for this assignment, larger state 5x5 is used. Moreover, there is a parameter P which user can modify to see different results. It will give variety and different results to the experiment. Lastly, the problem itself is very similar to the real life situation. Whether to cut the tree or save the tree has been ongoing ethical dilemma and I like to deal with those topics.

Second MDP used is a randomly generated MDP. It is always fun to deal with randomly generated values or matrix. This problem cosidered as having large states. First experiment has 1000 states with only 2 actions and second experiment has 1000 states with 10 actions. I found this problem interesting since I want to know how different algorithms work and give results with randomly generated MDPs. Sometimes, algorithms might work well with the given problem (where there is an answer to the problem) but not with the randomly generated matrix. This problem will be a good one to test the performances of each algorithms.

Experiment 1.1: Value iteration with forest management problem (S=5, p=0.1):

Value iteration uses Bellman Optimality Equation for computation. Bellman optimality equation is the interaction formula of optimal value functions. It simply changes the interaction formula to iterate and then, evaluate only once. Value iteration is not influenced by policy since it grabs the max value. It moves, computes the value, and give policy of 1 to the highest value. Therefore, this algorithm improves in a greedy way. So basically, value iteration selects value fuction, then iterate to find improved value function until it finds the optimal value function.

Value iteration for this assignment is done by using mdptoolbox. For the forest management problem, state of 5 (5x5) is chosen with default values for r1, r2, and p (r1=4, r2=2, p=0.1). For value iteration, parameter tuning is done for discount factor. Goal of hyperparameter tuing is to find the parameter that gives the same policy with less iterations. Discount factor is the present value of future rewards. It ranges from 0 to 1 and as it gets close to 1, it leads to "far-sighted" evaluation ("myopic" evaluation as it gets close to 0).

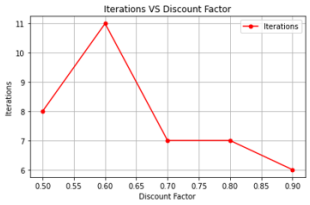


Figure 2: Iterations VS Discount Factor

Figure 2 shows the iterations taken to find optimal policy with different discount factors. Range of 0.5, 0.6, 0.7, 0.8, 0.9 is used. Discount factor of 0.9 had less iterations compare to the other values. Moreover for this assignment, I want to focus more on "far-sighted" evaluations.

Another parameter to describe is epsilon. Epsilon here is the stopping criterion. And it is the parameter we can choose between 0 to 1. If |V – V*| (V variance) is less than epsilon, it's considered as converged. Default of 0.01 is used and it is reasonable to use this value since it is a small number (we want to make the variance 0 or really close to 0). Figure 3 shows the variations of value functions in each iterations. As the number of iteration increases, variance decreases and at iteration 6, variance drops to zero. In other words, the value function converged with the optimal value function at iteration 6.

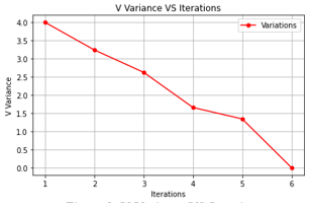


Figure 3: V Variance VS Iterations

Following table shows the results after applying tuned parameters. Value iteration found optimal policy of (wait, wait, wait, wait, wait) with 6 iterations within 0.0010 seconds.

	Policy	Iterations	Time
Value Iteration	(0, 0, 0, 0, 0)	6	0.0010

Experiment 1.2: Policy iteration with forest management problem (S=5, p=0.1):

Policy iteration is defined as process of updating the policy. With the Bellman Expectation Equation, policy iteration updates V with the given policy. (policy evaluation) Then, compute the Q value with the V value and Bellman expectation equation. It gives 1 to the action that maximizes the Q value and 0 to the rest. It iterates and updates the policy until the convergence. Convergence is achieved when the value doesn't change after improvement and evaluation. So basically, policy iteration first select policy and value function of that policy. Then it finds new policy based on the previous value function and iterate until it finds the optimal policy.

Policy iteration is also done by using mdptoolbox. Hyperparameter tuning is also for discount factor. Same range of discount factor is used for experiment purposes. (0.5, 0.6, 0.7, 0.8, 0.9) Discount factor of 0.5 and 0.6 found optimal policy within 4 iterations and 0.7, 0.8, 0.9 found optimal policy with 5 iterations. We can't conclude that discount factor of 0.5 and 0.6 had better results compare to the other 3 since it is the matter of where we want to focus. (whether to put weight on current or future) Discount factor of 0.9 will be used for this analysis to compare the results with the value iteration.

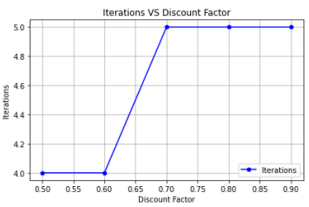


Figure 4: Iterations VS Discount Factor

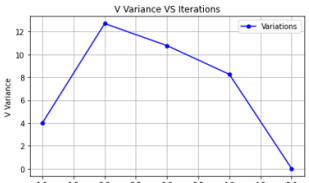


Figure 5: V Variance VS Iterations

Epsilon 0.01 is used for policy iteration. It is because 0.01 is small enough to talk about convergence and also, to keep parallel with the value iterations. For policy iteration, $|V_n - V_{policy}|$ should be less than epsilon to be considered as convergence. As shown on figure 5, the variation went up at start but decreased as the number of iteration increased. optimal epsilon policy is found at iteration 5.

Following is the table showing the results after applying tuned parameters.

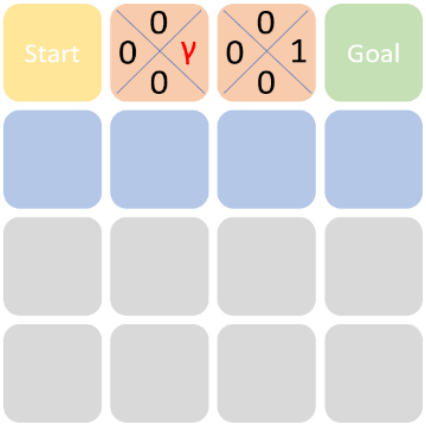
	Policy	Iterations	Time
Policy Iteration	(0, 0, 0, 0, 0)	5	0.0020

Experiment 1.3: Evaluation of Value Iteration & Policy Iteration:

Both value and policy iterations came up with the same optimal policy which is to (wait, wait, wait, wait, wait). Value iteration took 6 iterations and policy iteration took 5 iterations to converge. In terms of time, it took 0.0010 sec for value iteration to compute and 0.0020 sec for policy iteration to compute the optimal policy. Policy iteration took more time since the computation is expensive (solving linear system instead of just calculating Bellman operator).

Experiment 1.4: Q Learning with forest management problem (S=5, p=0.1):

Two algorithms above (value iteration and policy iteration) are model based algorithms. Q-learning algorithm is chosen for my favorite reinforcement learning algorithms. It is model free algorithm where the agent have no idea about transition probabilities or rewards. The goal of Q learning here is to learn optimal policy for agent to take actions under any circumstances. Q-learning uses the concept of exploration which is called epsilon greedy. It ranges from 0 to 1. For instance, there is a 4x4 grid (16 states). Blue square is the path of first episode. We started and reached the goal. However, it is clear that it is not the optimal path. Epsilon greedy randomly choose the path based on the values that we choose (0.5 = moves 50% randomly) so that it will not always take the same paths. High number of epsilon greedy might not be an optimal solution since exploration and exploitation might get unbalanced. Decaying epsilon greedy can be a good solution here. It is defined as decreasing the epsilon as it goes. In this case, it starts from 0.5 but decreases as it goes. This will have an effect of doing exploitation more and less exploration. Now, the problem is we are not sure which path is better. Orange path seems better than the blue one but there is no exact measurement to it since the reward is the same. To solve this we use the discount factor, also known as gamma. In this way, we can differenciate and rank the paths.



For Q-learning, there were several parameters to adjust in order to get the optimal policy. Those parameters are gamma, alpha, alpha min, alpha decay, epsilon, epsilon min, epsilon decay and iterations. It is good to tune all the parameters but for this assignment, gamma (discount factor), alpha (learning rate), and iterations is used. For both epsilon min and alpha min, 0.001 is used and the reason is because I wanted to maximize the range. Alpha decay and epsilon decay is set

to 0.01. In this way, I can examine the iterations and results in detail.

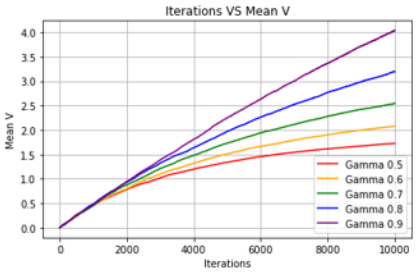


Figure 6: Iterations VS Mean V (Gamma)

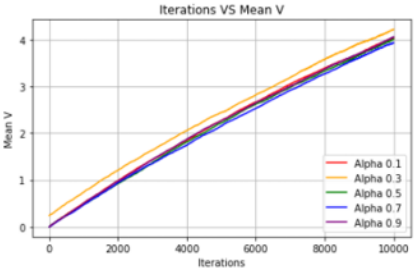


Figure 7: Iterations VS Mean V (Alpha)

Figure 8: Iterations VS Mean V (Iterations)

Different parameters are tuned to see the results of different exploration strategies. Figure 6 is the graph that shows how different gamma affect the mean V. Each usage has it's own pros and cons but for this assignment, I am following the parameter that has the highest mean V. Gamma 0.9 is used for the further analysis. Moreover, figure 7 shows the mean V after applying different alpha values (learning rate). The range is from 0.1~0.9 and 0.3 will be used. Figure 8 shows the mean V for different iterations. Iteration 10000 is not converged yet since the line is constantly going up. Iteration 100000 is close but still, there is no flat point. Iteration of a million and two million is converged. 2 Million iteration is used for future analysis.

Following is the table showing the results after applying tuned parameters.

	Policy	Iterations	Time
Q-Learning	(0, 0, 0, 0, 0)	Approx. 300,000	55.7135

Compare to value iteration & policy iteration, Q-learning found optimal policy with much more iterations and time. Approximately 300,000 iterations and 55.7135 seconds were needed to find the same policy. It was expected since value iteration and policy iteration are model based algorithms and Q-learning is considered as model free algorithm. But still, Q-learning was successful in getting the same policy as the other two.

Experiment 1.5: Modification of the forest problem (S=5, p=0.5):

Additionnal experiment was done to check if there are any changes after applying different conditions. For this experiment, number of state stayed the same but the probability of fire burn changed. I modified it from 0.1 to 0.5.

	Policy	Iterations	Time
Value Iteration	(0, 1, 0, 0, 0)	6	0.0025
Policy Iteration	(0, 1, 0, 0, 0)	4	0.0029
Q-Learning	(0, 1, 0, 0, 0)	Approx. 1,600,000	53.4725

Result was pretty much the same with the previous experiment. Both value and policy iterations gave the optimal policy of (wait, cut, wait, wait, wait). Value iteration took 6 iterations and policy iteration took 4 iterations to converge. Value iteration was a bit fast compare to the policy iteration but both algorithms were done very fast. For Q-learning, it took more iterations to converge compare to when p is 0.1. It is because there is more probability of a fire burn and this leads to more exploration needed to converge. But, Q-learning successfully found the optimal policy of (wait, cut, wait, wait, wait). Time it took (53.4725 sec) was similar to the previous experiment (55.7135 sec). Now, lets take a look at the problem with large number of states and see how different algorithms work with the problem.

Experiment 2.1: Random generated MDP problem with Value Iteration (S=1000, a=2):

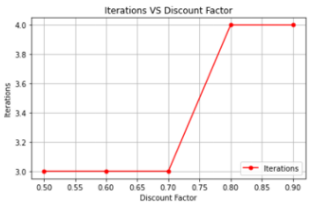


Figure 9: Iterations VS Discount Factor

Figure 9 shows the iteration taken to find optimal policy with different discount factors. Discount factor of 0.5, 0.6, 0.7, 0.8, 0.9 is used. 0.5, 0.6 and 0.7 found the optimal policy after 3 iterations and 0.8 and 0.9 found the policy after 4 iterations. In order to decide which discount factor to use, I took a look at the last variation V. Following table is the variation V of each iterations for each discount factor. Epsilon is set to default which is 0.01. The table shows that

Iteratiown \ \ Gamma	0.5	0.6	0.7	0.8	0.9
1	0.778425	0.778425	0.778425	0.778425	0.778425
2	0.020602	0.024722	0.028843	0.032963	0.037083
3	0.001412	0.002033	0.002768	0.003615	0.004575
4				**0.000260**	0.000314

Discount factor of 0.8 has the lowest variation which is close to nearly zero. Even it took extra iteration compare to other discount factor values, I want to focus more on far-sighted analysis.

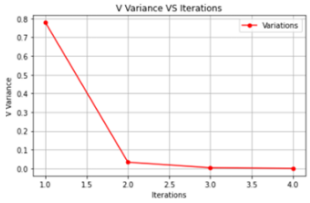


Figure 10: V Variance VS Iterations

Figure 10 shows and proves that the V variance drop below epsilon therefore found the optimal policy. Again, it is considered as convergence if |V – V*| (V variance) is less than the epsilon (0.01) in value iteration. As iteration increases, V variance decreased. At 4th iteration, V Variance became nearly zero. For this experiment, I tried to pick different epsilon values and compare the results. Epsilon of 0.1 had 3 iterations to find the policy, 0.01 with 4 iterations and 0.001 with 5 iterations. As iteration increased, the final iteration V variation got smaller. I decided to just use the default which is 0.01 since the v variation value is small enough (0.000260).

Following table shows the results after applying tuned parameters.

	Policy	Iterations	Time
Value Iteration	(0,1,1,1,1,0,1,0,0,1…0,0,1,0,1,0,1,0,1,0)	4	0.0049

Experiment 2.2: Random generated MDP problem with policy iteration (S=1000, a=2):

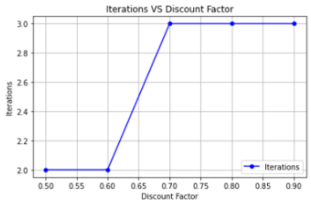


Figure 11: Iterations VS Discount Factor

Figure 11 shows the iteration taken to find optimal policy with different discount factors. Same with value iteration, discount factor of 0.5, 0.6, 0.7, 0.8, 0.9 is used. 0.5 and 0.7 found optimal policy after 2 iterations and other three found the optimal policy after 3 iterations. After looking at all the V variation values, 0.9 was chosen since it is far sighted and the variation was zero. Following table shows the V variation values of each iterations for various discount factors.

Iteratiown \ \ Gamma	0.5	0.6	0.7	0.8	0.9
1	0.778425	0.778425	0.778425	0.778425	0.778425
2	0.003493	0.004768	0.006045	0.007325	0.008606
3			0.000000	0.000013	**0.000000**

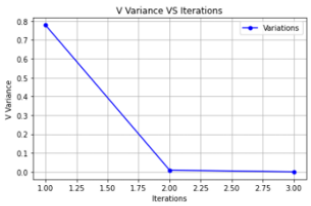


Figure 12: V Variance VS Iterations

Figure 12 shows that optimal policy is found after 3 iterations and it converged. For policy iteration, $|V_n - V_{policy}|$ should be less than epsilon to be considered as convergence. As iteration increases, V variance decreased and it reached 0 at after 3 iterations. Different epsilon is tested for policy iteration. When epsilon is 0.1, optimal policy was found within 2 iterations and 3 iterations for epsilon 0.01 and 0.001. I decided to go with epsilon 0.01 since it gave exact zero for variation.

Following table shows the results after applying tuned parameters.

	Policy	Iterations	Time
Policy Iteration	(0,1,1,1,1,0,1,0,0,1…0,0,1,0,1,0,1,0,1,0)	3	0.0229

Experiment 2.3: Evaluation of Value Iteration & Policy Iteration:

Both value and policy iterations came up with the exact same optimal policy (1000 policy actions were the same). Value iteration took 4 iterations and policy iteration took 3 iterations to converge. In terms of time, it took 0.0049 sec for value iteration to compute and 0.0229 sec for policy iteration to compute the optimal policy. Policy iteration took 5 times more in terms of time. Compare to the small state problem (forest management problem), it took a lot more. This is because state size is much bigger than the first problem. Interesting fact that I found here was the number of iteration took for value iteration and policy iteration to find optimal policy is actually less than the small state problem. This was very interesting since I thought if there are more states, it will take more iterations to find optimal policy regardless of the types of problem.

Experiment 2.4: Random generated MDP problem with Q-learning (S=1000, a=2):

For Q-learning, several parameters were adjusted in order to get the optimal policy. Those parameters are gamma, alpha, alpha min, alpha decay, epsilon, epsilon min, epsilon decay and iterations. After many experiments, I've decided to use alpha min and epsilon mean to 0.01 and alpha decay of 0.1, epsilon decay of 0.01 and epsilon of 0.1. Overall, the decay rate and values increased compare to the small forest management problem. It is because if we keep the same parameter, it takes very long and over 10 million iteration is needed.

For this assignment, gamma (discount factor), alpha (learning rate), and iterations is tuned.

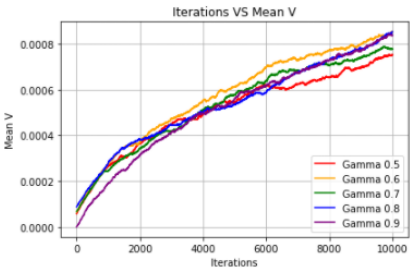


Figure 13: Iterations VS Mean V (Gamma)

Figure 14: Iterations VS Mean V (Alpha)

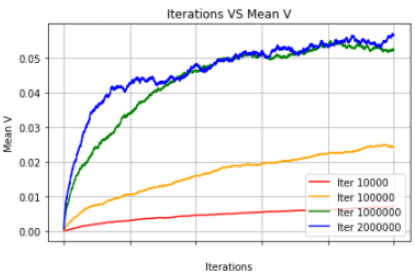


Figure 15: Iterations VS Mean V (Iterations)

Figure 13 shows the mean V with different gamma values. Range of 0.5, 0.6, 0.7, 0.8, 0.9 is used and gamma of 0.6 have the highest mean V in average. Similarly figure 14 shows the mean V after applying different alpha values. Range of 0.1, 0.3, 0.5, 0.7, 0.9 is used. The graph was not very consistent but after running several times, result of figure 14 was the average. Alpha value of 0.3 and 0.5 had better mean V compare to the other 3. Alpha of 0.3 is used. Figure 15 shows

the mean V with different number of iterations. Number of iterations used are 10000, 100000, 1 million, and 2 millions. Iteration 10000, 100000, and 1 million constantly goes up until the end of the iteration. There needs to be more iterations to make the line flat (convergence). Line of 2 million iterations goes up fast at start (approximately until it reaches 800000) but as number of iteration increases, it slows down. It still grows at the end of 2 million iterations but I could tell that the line is almost flat at the end. (converged)

Following table shows the results after applying tuned parameters.

	Policy	Iterations	Time
Q-Learning	(1,1,0,1,0,0,0,1,1,0…1,1,0,1,1,0,1,1,0,1)	Approx. 2 millions	597.2913

Compare to value iteration & policy iteration, Q-learning found optimal policy with much more iterations and time. Approximately 2 million iterations and 597.2913 seconds were needed to find the optimal policy. Interesting fact here is that the optimal policy was not the same with value & policy iteration. It might be because Q-learning algorithm does not know the model or the reward at start. Trying more number of iterations might end up finding the same optimal policy but it is uncertain that it will have the same policy with value and policy iterations.

Experiment 2.5: Modified random mdp problem (S=1000, a=10):

Additional experiment was done to check if there are any changes after applying different conditions. For this experiment, number of state stayed the same but the number of action changed from 2 to 10.

	Policy	Iterations	Time
Value Iteration	(4,2,2,6,8,9,4,0,9,7…7,8,1,4,8,6,4,3,7,4)	5	0.0179
Policy Iteration	(4,2,2,6,8,9,4,0,9,7…7,8,1,4,8,6,4,3,7,4)	3	0.0229
Q-Learning	(4,6,2,0,8,0,4,2,7,8…5,2,1,8,0,3,8,0,7,4)	Approx. 2 millions	658.268

Result was pretty much the same with the previous experiment. Both value and policy iterations gave the same optimal policy. Value iteration took 5 iterations and policy iteration took 3 iterations to converge. Value iteration was a bit fast (0.0179 sec) compare to the policy iteration (0.0229) but both algorithms were done very fast. For Q-learning, the result was about the same. Optimal policy was different from the other two (value iteration and policy iteration) and also, took much longer time.

Conclusion:

There was a certain difference between model-based algorithms and model-free algorithms. After the experiment, it seems like model-based algorithms worked better compare to the model-free algorithms but there are advantages and disadvantages to it. Since model-based algorithms utilize the existing knowledge (model, rewards…etc), it could do a productive learning with small amount of data. Model-free algorithms can be used even it is hard to set up a model for the environment. Overall, all the algorithms had it's own strengths and weaknessess. There was lots of learning curves going on experimenting those algorithms to solve small and large MDPs.

Citation:

Cordwell, Steven A W. “Markov Decision Process (MDP) Toolbox: Example Module¶.” *Markov Decision Process (MDP) Toolbox: Example Module - Python Markov Decision Process Toolbox 4.0-b4 Documentation*, 2015, pymdptoolbox.readthedocs.io/en/latest/api/example.html.

Cordwell, Steven A W. “Markov Decision Process (MDP) Toolbox: Mdp Module¶.” *Markov Decision Process (MDP) Toolbox: Mdp Module - Python Markov Decision Process Toolbox 4.0-b4 Documentation*, 2015, pymdptoolbox.readthedocs.io/en/latest/api/mdp.html.

Hiive. “Hiive/Hiivemdptoolbox.” *GitHub*, 25 Nov. 2019, github.com/hiive/hiivemdptoolbox/blob/master/hiive/mdptoolbox/mdp.py.