Markov Decision Processes

Daniel Bach
dbach7@gatech.edu

1 MARKOV DECISION PROCESSES

Two Markov decision processes (MDP) will be examined in this analysis. The first being blackjack and the second being the taxi problem. Both problems are publicly available through the Farama Foundation's Gymnasium library [1]. Three different approaches to solving MDPs will be used: value iteration, policy iteration, and the Q-learning algorithm.

1.1 Blackjack

Blackjack is a common card game that appears in many casinos. This problem is interesting as it is an example of a real game played with a reward everyone is familiar with: money.

In this analysis, the rules outlined from Sutton and Barto's example 5.1 will be used [2], [3]. The cards are drawn from an infinite deck with replacement. If a player's first two cards sum up to 21 (via an ace and a 10 or face card), this is called a natural and the player wins immediately unless the dealer also has a natural. If the dealer also has a natural, then the game is a tie.

Defining blackjack in MDP terms: the state space (S) is made up of a tuple: (player's current sum, dealer's face up card, player holds a usable ace). There are 290 possible states. The action space (A) is either 0 (stick) or 1 (hit). The rewards (R) can either be +1 (win), 0 (tie), or -1 (lose). The transition probabilities (P) can be thought of as the probability of increasing the player's hand value to a higher value if they hit. Of course, the probability of the number changing is 0.00 when the player chooses to stick.

1.2 Taxi

Taxi is a grid world problem where the taxi is able to navigate around, pick up, and drop off a passenger [4]. The passenger starts at one of the locations designated by a blue, yellow, red, or green square and must be picked up and dropped off at another colored square. This problem is much more challenging than the blackjack problem due to the increased number of spaces and actions.

There are 25 possible taxi locations, 5 possible passenger locations (including the passenger being in the taxi), and 4 destination locations [4]. Defining this problem in MDP terms: the state space (S) is defined as one number which is calculated by the

formula: ((taxi_row * 5 + taxi_col) * 5 + passenger_location) * 4 + destination [4].There are 400 reachable states which can be reached during an episode, with an additional 4 if the taxi is successful and drops off the passenger at the specified location. The action space (A) consists of 6 acts: the taxi moving left, right, up, down, and picking up and dropping off the passenger. There are 3 rewards (R): -1 at each step unless another reward is triggered, +20 for delivering the passenger successfully, and -10 for any illegal pick up or drop off action.

2 BLACKJACK

When performing the value iteration and policy iteration algorithms on the blackjack MDP, a discount factor of γ=1.0 was used. In the blackjack problem, the player aims to win each individual episode, and there are no advantages of winning one round over another since the reward doesn't change from episode to episode and the reward is only obtained at the end of the episode [5].

2.1 Value iteration

The first experiment performed when performing value iteration was to change the gamma parameter to see if the results are what is expected from the reasoning before. The bettermdptool's library [6] allows for three parameters for its value iteration function: gamma, n_iters, and theta. n_iters is the maximum number of iterations the function will perform to attempt to find the optimal value function. theta is the converge threshold criterion for the maximum difference between the new and different state values [6]. For this experiment, n_iters was left to its default value of 1,000 and theta was left to its default value of 1e-10.

Table 1—Blackjack results for changing the gamma parameter for value iteration

gamma(γ)	**Iterations to Converge**	**Average Reward**	**Win % (over 100,000 Episodes)**
0.1	8	-0.0461	43.452
0.3	9	-0.0440	43.475
0.5	10	-0.0460	43.458
0.6	11	-0.0494	43.211
0.7	11	-0.0422	43.401
0.8	11	-0.0399	43.538
0.9	12	-0.0434	43.397
1.0	12	-0.0394	43.501

To obtain the iterations to converge, V_track (returned by the value iteration function), which is a log of the value function V(s) for each iteration was inspected to see when there were no further updates. The average reward and win % was calculated from statistics of playing the 100,000 episodes using the policy π output by the value iteration function.

The results are close to what is expected, although the relationship might not be quite linear due to variance. The average reward tends to increase with higher values of gamma. gamma should be kept closer to 1.0 for the blackjack problem as each episode is independent of each other and rewards only come at the end of the episode.

2.2 Policy iteration

To compare policy iteration against value iteration, both were left to their default parameters. bettermdptool's policy iteration function takes the same parameters as the value iteration function does. The only difference is that for policy iteration, the default n_iters=50 vs. 1,000 for value iteration.

Table 2—Blackjack results using the policy from policy iteration and value iteration

Algorithm	Runtime (s)	Iterations to Converge	Reward	Win % (over 100,000 Episodes)
Policy Iteration	0.322	4.0	-0.0443	43.218
Value Iteration	0.0212	12.0	-0.0419	43.372

These results were calculated by taking the average results from 5 trials. Interestingly, even though policy iteration takes less iterations to converge, the runtime of the algorithm is about ~15x longer. This could be due to the separation of the policy evaluation and improvement steps within the algorithm. For value iteration, these steps are combined. The policy output from value iteration was able to perform slightly better. The results are still close however, and that is expected since value iteration and policy iteration are both expected to converge to the optimal policy, but using different methods [5].

2.3 Q-Learning

The Q-learning reinforcement learning algorithm was applied to the blackjack problem. bettermdptools also provides a Q-learning function available to use at [7] with a wide variety of parameters. An experiment was performed to see how the policy output by the Q-learning algorithm would perform as the number of episodes were changed as the algorithm does not have a built-in early termination criterion.

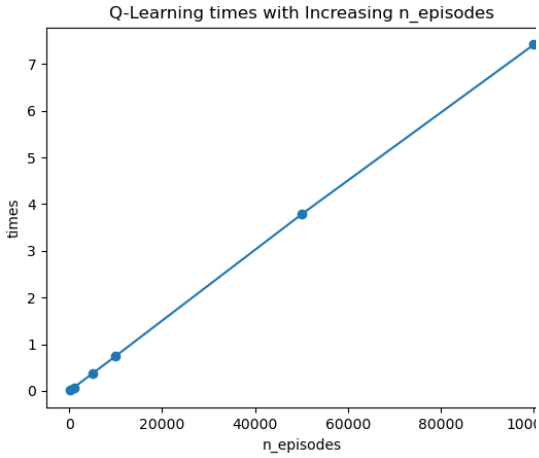

Figure 1 **-** Runtimes for Q-Learning on blackjack

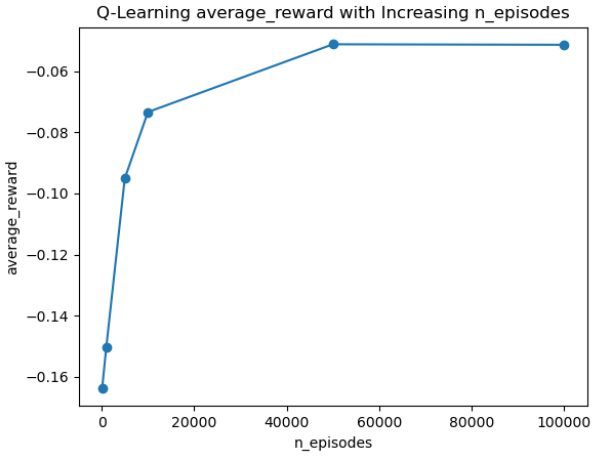

Figure 2 **-** Average reward for Q-Learning

These results were averaged across three trials for each number of n_episode. The n_episodes parameter for the Q-learning algorithm was varied from: [100, 1,000, 5,000, 10,000, 50,000, 100,000]. For reference, the default parameter is 10,000. The runtime scales nearly linearly, this makes sense as the operations within the Q-learning algorithm do not change depending on n_episodes, but the algorithm will go through more iterations of the same calculations. Diminishing returns on the average blackjack reward was seen around n_episodes=50,000 as there is no improvement in average rewards from 50,000 to 100,000 n_episodes.

2.4 Blackjack results

Comparing policy and value iteration, value iteration was more performant as it was able to converge to an optimal value function ~15x faster than policy iteration while gaining slightly more reward. It is also shown that policy and value iteration outperform Q-learning when looking at the runtimes and average reward from each episode using the policies outputted by each algorithm. This is most likely due to the complete knowledge (such as the probabilities transition matrix, P) of the blackjack environment which was passed into the policy and value iteration algorithm. Due to the small state space and non-complex nature of the blackjack MDP problem, value and policy iteration is best suited over Q-learning.

3 TAXI

In this section, the three former algorithms will be tested on a larger MDP. To reiterate, the taxi problem contains 404 reachable states while Blackjack had 290 possible states. There are also 6 total actions that the agent can take with rewards that are given throughout the episode and not just at the end.

3.1 Value and policy iteration

In the first experiment for the taxi problem, the optimal value of the gamma parameter for the value iteration and policy iteration algorithms was searched for. gamma is a more important parameter to consider in the taxi problem as there are penalty rewards that may be given at each step. The only positive reward comes from picking up and dropping the passenger off successfully so the future reward is the one of most importance.

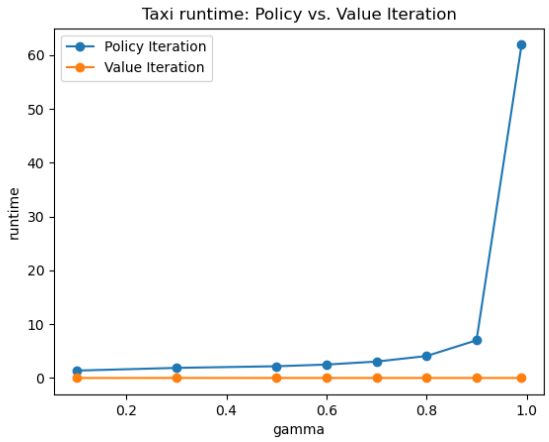

Figure 3 - Runtimes for the taxi problem

Figure 4 - Average reward over 10 episodes

One of the results that stands out the most from this experiment is how much gamma affects the runtime for the policy iteration algorithm. At gamma=0.99, policy iteration took 61.92 seconds to converge. This is due to higher values of gamma causing the algorithm to put more emphasis on long-term rewards. The influence of the long-term rewards take longer to propagate throughout the value iteration algorithm [5].

Surprisingly, the average reward over 10 episodes for each gamma did not vary much after gamma=0.1 for the value iteration algorithm, which was the only case where the algorithm did not always successfully pick up and drop off the passenger. With small gamma, the goal would be to minimize the negative rewards at each step, so the policy would be to pick up and drop off the passenger in as few moves as possible.

3.2 Q-Learning

For the taxi problem, several experiments were performed on the hyperparameters to see how each one would affect the Q-learning algorithm. The number of iterations were dropped from its default value of 10,000 to 500 so that the effects of each hyperparameter were more clear. Maximum Q-values were inspected, which is the maximum reward the agent could achieve in a state, s [5].

3.2.1 Gamma

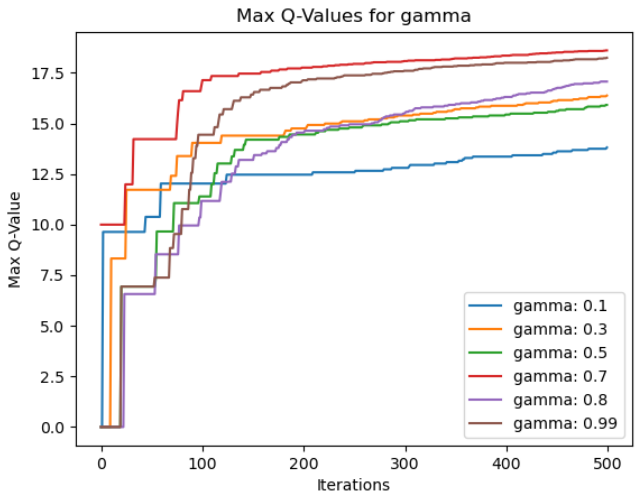

Figure 5 - Max Q-Values at each iteration for different gamma values

The gamma hyperparameter is still the discount factor in regards to the Q-learning algorithm. When gamma=0.1, the maximum Q-value was at its lowest, which was 12.5. This could be due to the fact that the algorithm does not give much importance to future rewards so the agent could just be minimizing its actions of illegally dropping off and picking up passengers, which is a -10 reward, whereas just moving is a -1 reward

3.2.2 Alpha

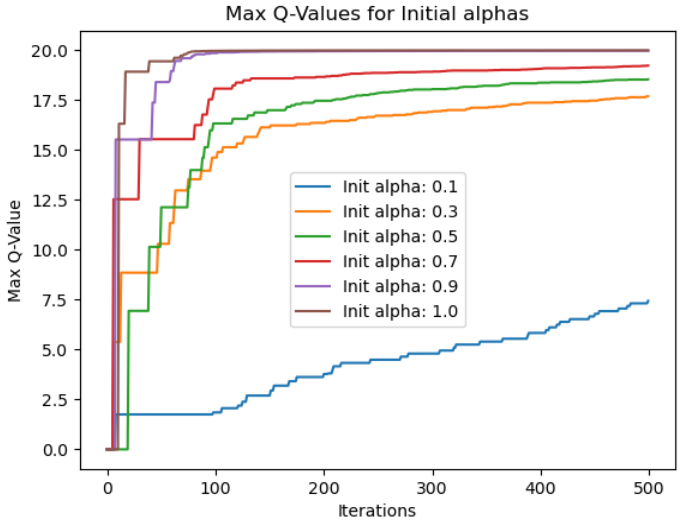

Figure 6 - Max Q-Values at each iteration for different initial alpha values

The alpha parameter is the learning rate, which determines how much the Q-values are updated at each iteration. A higher alpha could lead to more volatile learning, whereas a low alpha could result in slower convergence [5]. This effect is seen clearly in the graph, as when alpha is initialized to 0.1, the maximum Q-value increases very slowly compared to all other values of alpha. The fastest increase in the max Q-value is seen with alpha=1.0, which reaches the maximum reward of 20 first.

3.2.3 Epsilon

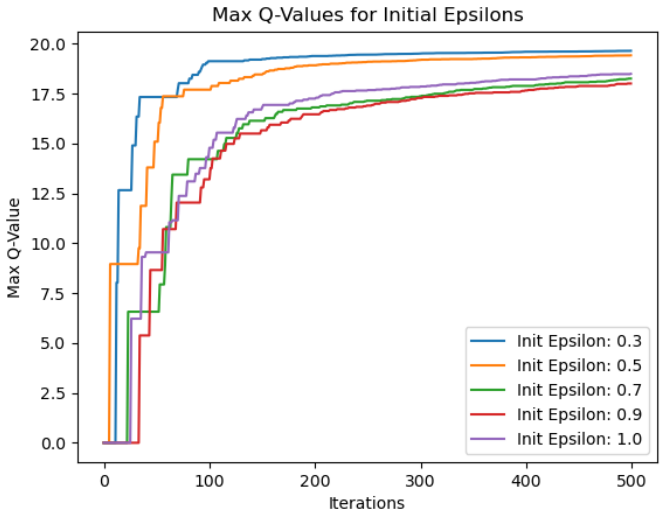

Figure 7 - Max Q-Values at each iteration for different initial epsilon values

The epsilon parameter is the exploration rate. If epsilon is higher, then the Q-learning agent is more likely to select a random action instead of the action predicted by the current Q-values. Looking at the graph, it is seen that the agent performed the best with a lower initialized epsilon, which follows a greedy approach of sticking with the best action instead of exploring. Higher values of epsilon might not perform well initially (since 500 is a low number of iterations) because they might try actions such as illegally dropping off and picking up the passenger which is a -10 reward.

3.3 Taxi conclusion

Similar to the blackjack MDP, greedy approaches such as value iteration and policy iteration performed well. Different hyperparameters which affect the Q-learning process were explored. High values of gamma performed the best, indicating that the agent should give higher weight to long-term rewards. A high learning rate, alpha, helps the Q-learning agent converge faster towards the maximum reward. Lastly,

epsilon should be low to make the Q-learning algorithm more greedy and similar to the value iteration and policy iteration algorithms.

4 REFERENCES

1. https://github.com/Farama-Foundation/Gymnasium (accessed Nov. 22, 2023)
2. R. Sutton and A. Sarto., *Reinforcement Learning: An Introduction*. Online, 2020. http://www.incompleteideas.net/book/RLbook2020.pdf
3. https://gymnasium.farama.org/environments/toy_text/blackjack/#observation-space (accessed Nov. 22, 2023).
4. https://gymnasium.farama.org/environments/toy_text/taxi/
5. https://chat.openai.com/share/129aa869-b173-4fe2-915d-fc24d308eb76 (Accessed Nov. 26, 2023)
6. https://github.com/jlm429/bettermdptools/tree/master (accessed Nov. 22, 2023)
7. https://github.com/jlm429/bettermdptools/blob/master/algorithms/rl.py (accessed Nov. 23, 2023)