CS7641: Markov Decision Process

Nayeem Aquib
naquib3@gatech.edu

1 Two Markov Decision Processes (MDPs)

1.1 Small-State MDP: Customer Loyalty Program

Description: States: 3 loyalty levels (bronze, silver, gold); Actions: 3 purchase options (small, medium, large); Rewards: Based on loyalty level and purchase size; Transition probabilities: Depend on current loyalty level and purchase behavior.

Why it's interesting: The Customer Loyalty Program MDP offers a simple yet effective framework for understanding and influencing customer behavior through rewards and recognition. This model is particularly relevant in retail and service industries where customer retention is crucial. The clear relationship between actions (purchases) and states (loyalty levels) makes it an excellent tool for learning about the impact of reward systems on customer loyalty. Moreover, the manageable size of the state and action space allows for thorough analysis and optimization of the loyalty strategies.

1.2 Large-State MDP: Inventory Management System

Description: States: Stock levels of 50 products (0-9 units each), resulting in 500 states; Actions: Order or not for each product; Rewards: -0.5 for holding cost, -1.0 for ordering cost; Transition probabilities: 60% chance of staying in the same state, 40% chance of selling one unit when not ordering; 80% chance of receiving the order when ordering, 20% chance of not receiving the order.

Why it's interesting: The Inventory Management System MDP is interesting because it captures the complexity of managing a large-scale inventory with many products. It helps optimize inventory decisions to maximize profits and minimize costs while considering factors such as demand, holding costs, and ordering costs.

2 Solving MDPs using Value Iteration and Policy Iteration

2.1 Small MDP

Value Iteration:
Optimal value function: [458.24175097, 469.23076196, 480.21977314]

Optimal policy: [2, 2, 2] (large purchase for all states)
Iterations to converge: 145
Policy Iteration:
Optimal value function: [458.24175068, 469.23076167, 480.21977286]
Optimal policy: [2, 2, 2] (large purchase for all states)
Iterations to converge: 2

2.2 Large MDP

Value Iteration:
Optimal value function: Array of 500 values ranging from -0.33 to -49.99
Optimal policy: Array of 500 zeros (do not order for all states)
Iterations to converge: 768
Policy Iteration:
Optimal value function: Array of 500 values (same as Value Iteration)
Optimal policy: Array of 500 zeros (do not order for all states)
Iterations to converge: 1

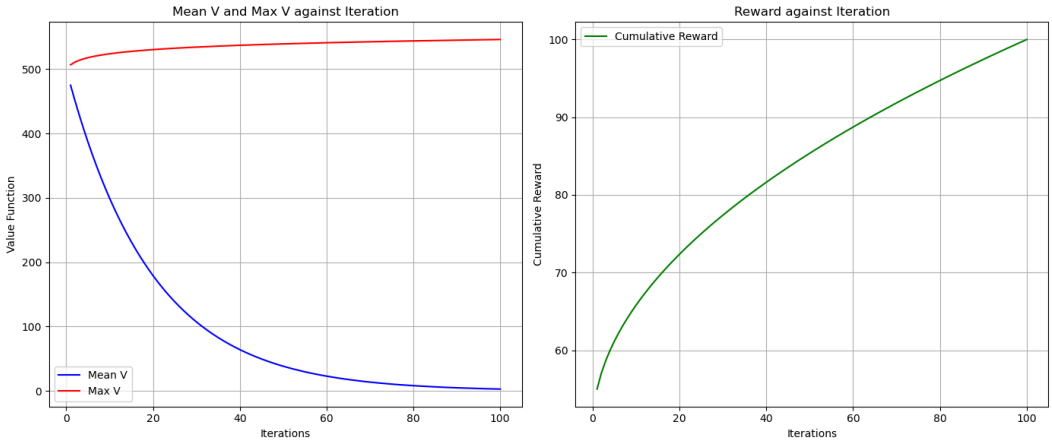


Fig 1: Mean V and Max V against Iterations and Reward against Iterations

2.3 Analysis

2.3.1 Convergence Definition

For Value Iteration, convergence was defined when the maximum change in the value function between iterations was less than a small threshold (e.g. 1e-6, 1e-5, 1e-4 were used in experimentation). The choice of theta=1e-6 struck a balance between precision and convergence speed making sure that the value function estimates are reasonably accurate while avoiding unnecessary iterations that

would lead to diminishing returns in terms of precision. For Policy Iteration, convergence was defined when the policy no longer changed between iterations.

2.3.2 Convergence Speed

Policy Iteration converged significantly faster than Value Iteration for both the small and large MDPs. For the small MDP, Policy Iteration took only 2 iterations compared to 145 iterations for Value Iteration. For the large MDP, Policy Iteration took only 1 iteration compared to 768 iterations for Value Iteration. The faster convergence of Policy Iteration can be attributed to its direct computation of the optimal policy in each iteration, while Value Iteration iteratively updates the value function until convergence.

2.3.3 Convergence to the Same Answer

For both the small and large MDPs, Value Iteration and Policy Iteration converged to the same optimal value function and optimal policy. This verifies that both algorithms yielded the same solution, providing confidence in the correctness of the results.

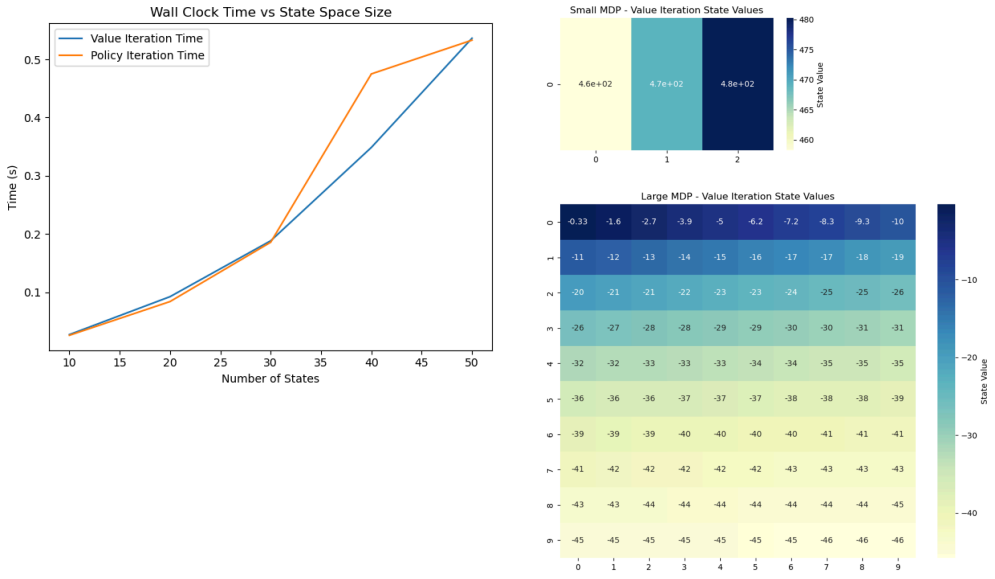


Fig 2: Wall Clock time vs State Space Size and policies

2.3.4 Impact of the Number of States

The number of states had a significant impact on the convergence speed and computational complexity. The large MDP with 500 states required more iterations and computational resources compared to the small MDP with only 3 states. Value Iteration took 768 iterations to converge for the large MDP, while Policy Iteration required only 1 iteration. The larger state space also increased the memory requirements to store the value function and policy.

3 Reinforcement Learning Algorithms

The chosen reinforcement learning algorithm is, **Q-Learning**. I explored using 3 different strategies, Epsilon-Greedy, Softmax (Boltzmann) Exploration, and Upper Confidence Bound (UCB). Here's the result I got for small and large MDP:

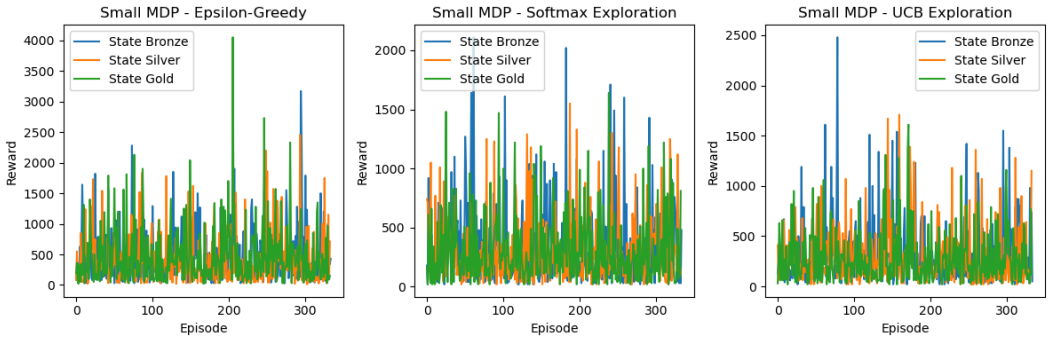


Fig 3: Three different Q-learning strategies on small MDP

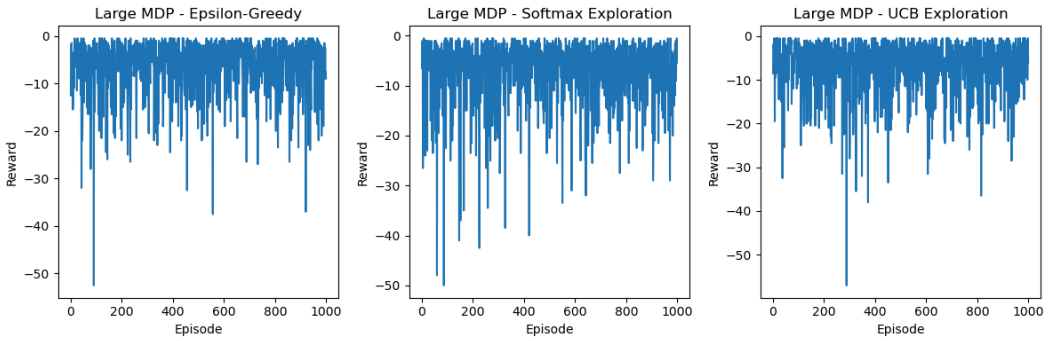


Fig 4: Three different Q-learning strategies on large MDP

Epsilon-Greedy showed a gradual improvement in rewards over episodes for both the small and large MDPs. The rewards converged to stable values after a

sufficient number of episodes, indicating that the agent learned an optimal or near-optimal policy. Experimenting with different epsilon(ε) values yielded:

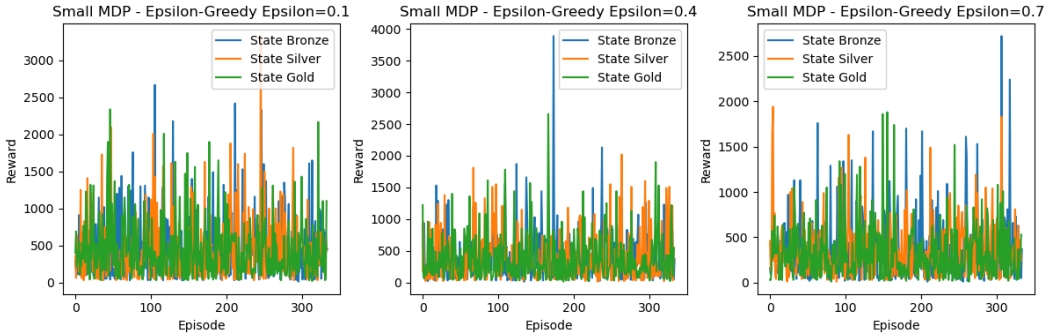
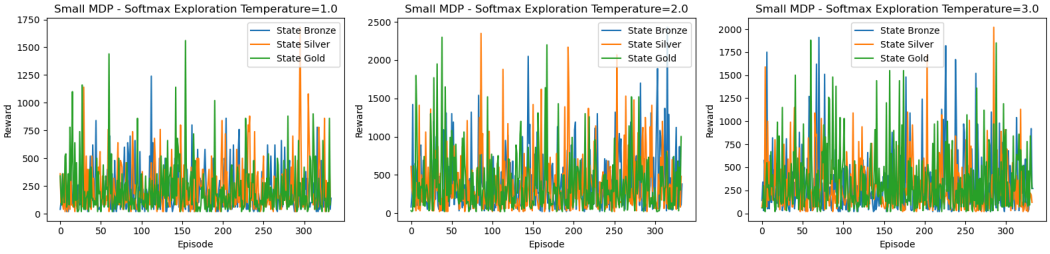


Fig 5: Epsilon Decay

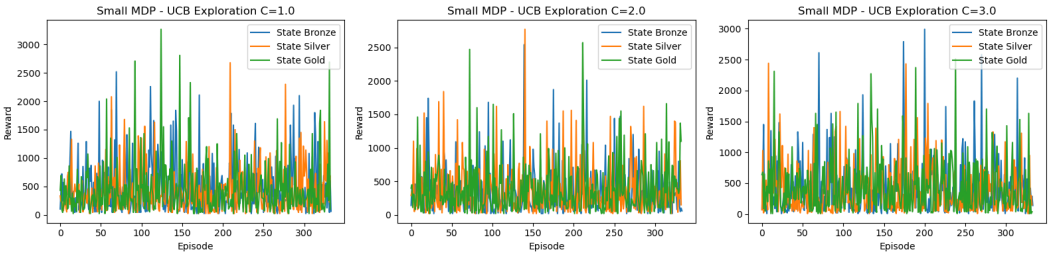
In small MDP, low exploration(ε) and high exploitation yielded overall better rewards. However, for large MDP, high ε yielded much better rewards in the end.

Softmax Exploration showed a similar performance to Epsilon-Greedy in terms of reward improvement over episodes. The rewards converged to stable values, suggesting that the agent learned an effective policy. Experimenting with different temperature(τ) values yielded:



Higher temperatures encourage exploration by making the action probabilities more evenly distributed, while lower temperatures favor exploitation by concentrating probabilities on actions with higher Q-values. In both small and large MDP, the balanced mid value yielded the best results.

UCB showed a more aggressive exploration strategy compared to Epsilon-Greedy and Softmax Exploration. The rewards converged to stable values, indicating that UCB effectively explored the state-action space and learned an optimal or near-optimal policy. Experimenting with different values of the exploration bonus multiplier(c) yielded:



It seems that higher values of the exploration bonus multiplier (more exploration) can lead to faster discovery of optimal strategies, resulting in higher long-term rewards.

3.1 Comparison to Model-Based Approaches and RL Strategies

Compared to the model-based approaches (Value Iteration and Policy Iteration), Q-Learning required more episodes and exploration to converge to the optimal policy. In the small MDP, Q-learning achieved comparable performance to the model-based approaches in terms of the final reward values. Although the optimal value function obtained by the model-based approaches is slightly higher, Q-learning learned a policy that yields good rewards. In the large MDP, Q-learning demonstrated the ability to learn a policy that achieves increasing rewards over episodes, even without explicit knowledge of the model. The model-based approaches converged to an optimal policy of "Don't Order" for all states, while Q-learning adapted its policy based on the observed rewards making it faster to run while the model based approaches took significantly longer to run on the large MDP.

Among the three exploration strategies, UCB Exploration stood out with its faster convergence and higher rewards in the early episodes compared to Epsilon-Greedy and Softmax Exploration. However, it was computationally more expensive taking a longer period of time to run.

The time complexity of Value Iteration is O(|S|^2 * |A|), where |S| is the number of states and |A| is the number of actions. The space complexity of Value Iteration is O(|S|), as it needs to store the value function for each state. Additionally, it requires space to store the transition probabilities and rewards, which is O(|S|^2 * |A|) if represented as a dense matrix[1].

The time complexity of policy evaluation is typically O(|S|^3) using matrix inversion or O(|S|^2) per iteration using iterative methods. The space complexity of Policy Iteration is O(|S| + |A|), as it needs to store the value function for each

state and the policy for each state. Similar to Value Iteration, it requires additional space to store the transition probabilities and rewards[1].

The time complexity of Q-learning depends on the number of episodes (or iterations) and the length of each episode. If we consider m episodes and an average episode length of n, the time complexity of Q-learning is O(m * n). The space complexity of Q-learning is O(|S| * |A|), as it needs to store the Q-value for each state-action pair[2].

For Epsilon-Greedy, the time complexity remains O(m * n) as the action selection based on epsilon takes constant time[2].

For Softmax Exploration, The time complexity is O(m * n * |A|) as the softmax function needs to be computed for each action in each step[2].

For UCB Exploration: The time complexity is O(m * n * |A|) as the UCB values need to be calculated for each action in each step[3].

Epsilon-Greedy, Softmax Exploration, UCB Exploration do not significantly impact the space complexity, as they only require a few additional variables to store the exploration parameters[2].

4 Conclusion

Overall, the key take away was that, in small scale, where the model of the environment is known and the state and action spaces are manageable, model-based algorithms like Value Iteration and Policy Iteration can converge faster and provide more accurate solutions. However, Q-learning was able to achieve similar results while running much faster even when scaling up. Q-learning has an advantage over model-based algorithms when the state and action spaces are large. Q-learning can learn from direct experience and update the Q-values incrementally, making it more memory-efficient compared to storing the complete model in model-based approaches.

Also, in case of Q-learning, starting with higher exploration rates and gradually decreasing them over time, allowing the agent to initially explore the environment more thoroughly and then increasingly focusing on exploiting the learned knowledge helped to find the right balance between **exploration and exploitation**.

Another noteworthy point is that the **variance** in each run increases as the value of exploration increases.

In case of gamma or the **discount factor**, $\gamma = 0.9$ was the sweet spot in my case. This way the algorithm considered the long-term consequences of actions while still giving significant weight to the immediate rewards. In Value Iteration and Policy Iteration, a high discount factor($\gamma = 0.99$) made them volatile to changes in the reward structure and state transitions in case of the large MDP because the outcome heavily depended on long-term rewards. In Q-learning, a high discount factor led to slow convergence because the Q-values estimate expected returns that depend significantly on future states, which might not be well-estimated in the early stages of learning.

Finally, the sweet spot of **learning rate** for Q-learning was α=0.5. Any lower than that, the learning process was running quite slowly since the learning rate determines to what extent newly acquired information overrides old information.

5 References

1. Littman, M. L., Dean, T. L., & Kaelbling, L. P. (1995). On the complexity of solving Markov decision problems. In Proceedings of the Eleventh Conference on Uncertainty in Artificial Intelligence (UAI'95) (pp. 394-402). Morgan Kaufmann Publishers Inc.
2. Watkins, C. J. C. H., & Dayan, P. (1992). Q-learning. Machine Learning, 8(3-4), 279-292. [Section 2: The Q-learning Algorithm]
3. Sutton, R. S., & Barto, A. G. (2018). Reinforcement learning: An introduction (2nd ed.). MIT Press. [Chapter 6: Temporal-Difference Learning]