Reinforcement Learning: Assignment 2

Course: DSCI 6650-01: Reinforcement Learning
Student: Sahil Khan
Date: July 25, 2025

Introduction

This report details the application of various reinforcement learning algorithms to solve a 5x5 gridworld problem. The objective is to find optimal policies for an agent navigating the grid under two different environmental configurations. Part 1 focuses on Dynamic Programming (DP) methods where the model of the environment is known. Part 2 explores model-free Monte Carlo (MC) methods in a modified, episodic version of the environment. The analysis includes the calculation of value functions and the determination of optimal policies, supported by visual plots and discussion of the results.

Part 1: Dynamic Programming in a Gridworld

The first part of the assignment considers a 5x5 gridworld with four special states (Blue, Green, Red, Yellow) that provide specific rewards and cause deterministic or stochastic transitions. The agent can move up, down, left, or right.

1.1 Policy Evaluation

Methodology:
The value function for a fixed, random policy (where each action has a 0.25 probability) was estimated using two methods with a discount factor of γ=0.95:

1. **Direct Solution:** The system of 25 simultaneous Bellman equations was constructed in matrix form (I - γP)V = R and solved directly for the value function vector V.
2. **Iterative Policy Evaluation:** The value function was initialized to zero and iteratively updated using the Bellman expectation equation until the change in value for any state was below a small threshold (1e-6).

Results:
Both methods produced identical value functions, validating the correctness of the implementations. The iterative method converged in 191 iterations.

Analysis:
It was interesting to see that both methods gave the exact same values for each square. This was a good check to confirm the math was right.

At first, it was surprising that the most valuable spots were at the top of the grid, far from the red and yellow squares. But it makes sense because the blue square, with its big +5 reward, is right there. Even with a totally random policy, any square next to the blue one has a 1-in-4 shot at a huge payoff on the next move, which really pumps up its long-term value.

1.2 Optimal Policy Search

Methodology:
The optimal policy was determined using two primary DP algorithms:

1. **Policy Iteration:** This method alternates between two steps: evaluating the current policy (Policy Evaluation) and then improving the policy by acting greedily with respect to the new value function. This process was repeated until the policy stabilized.
2. **Value Iteration:** This method directly finds the optimal value function by iteratively applying the Bellman optimality equation update. The optimal policy is then extracted by finding the action that maximizes the expected value from each state.

Results:
Both Policy Iteration and Value Iteration converged to the same optimal value function and optimal policy. Policy Iteration stabilized in 5 iterations, while Value Iteration converged in 302 iterations.

Analysis:
Both Policy Iteration and Value Iteration found the same best strategy, which is what I expected. The optimal plan is simple: no matter where you are, head for the blue square as fast as possible. It gives the best reward and then teleports you to the red square, where you just start heading back to the blue one again.

The agent learns to pretty much ignore the green square because its reward isn't as good and the teleport is a gamble. It was also interesting to note that Value Iteration was much faster to run, since each of its update steps is simpler than the full evaluation step required in Policy Iteration.

Part 2: Monte Carlo Methods

The environment was modified to be episodic by adding three terminal (black) states. The reward for any standard move was changed to -0.2 to encourage shorter paths.

2.1 On-Policy Monte Carlo Control

Methodology:
The optimal policy was learned using two on-policy MC methods, which learn from episodes generated by the same policy being improved.

1. **MC with Exploring Starts:** To ensure all state-action pairs were visited, each episode was started with a randomly chosen state and a randomly chosen first action.
2. **MC with ε-soft Policy:** A more practical approach where the agent follows a greedy policy most of the time but explores a random action with a small probability (ε=0.1).

Results:
Both methods successfully learned a near-optimal policy. The Exploring Starts method used 50,000 episodes, while the ε-soft method used 100,000 episodes to achieve a comparable result.

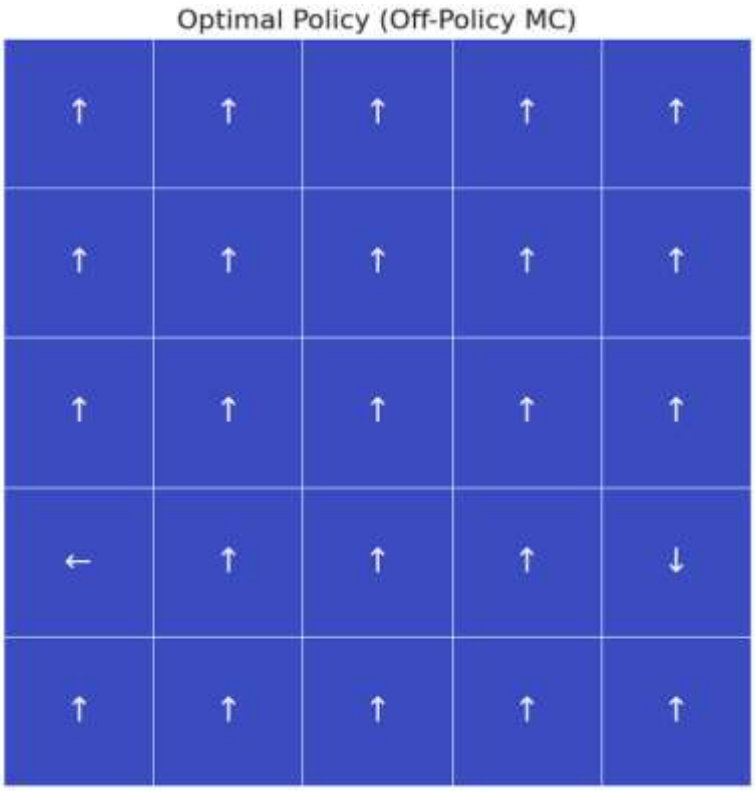
Analysis:
The Monte Carlo methods learned a similar "go for the blue square" strategy, but what's neat is that they did it without knowing the rules of the game beforehand. They learned just by trying things out and seeing what happened over thousands of episodes.

The "Exploring Starts" method worked well but seems a bit unrealistic since you can't always start an episode wherever you want in a real problem. The epsilon-soft approach is more practical. It found a great policy, though it might not be 100% perfect because it always keeps a small chance (epsilon) of exploring a random move instead of what it thinks is the best one. The final policy also clearly shows the agent learning to stay away from the black terminal squares, since they offer no path to a reward.

2.2 Off-Policy Monte Carlo Control

Methodology:
An off-policy approach was used to learn the optimal (target) policy while following a different, more exploratory (behavior) policy. The behavior policy was set to choose actions randomly with equal probability. Weighted importance sampling was used to correct for the difference between the policies, allowing the agent to learn about the optimal greedy strategy from the random episodes.

Results:
This method also converged to the optimal policy after 200,000 episodes. The resulting policy is visually identical to those found by the on-policy and DP methods.



Analysis:
Off-policy learning was really interesting. It allowed the agent to learn the best way to do things, even while it was behaving totally randomly to explore the grid. This is powerful because you can learn the perfect strategy without actually having to follow it during the learning process.
The downside was that it took way more episodes (200,000) to get a good result compared to the on-policy methods. This is because of the "importance sampling" weights, which are used to correct for the fact that the agent's actions (random) are different from the policy it's trying to learn (greedy). These weights can sometimes get really big, which makes the learning a bit less stable and slower. Still, it successfully found the optimal policy in the end.

Conclusion

This assignment successfully demonstrated the application and comparison of Dynamic Programming and Monte Carlo methods. DP methods (Policy and Value Iteration) are effective and efficient when a model of the environment is available, converging quickly to the exact optimal policy. MC methods (On-Policy and Off-Policy) provide a powerful model-free alternative, learning directly from experience. While they require more samples to converge, they successfully identified the same optimal strategies as their DP counterparts, highlighting their utility in more complex, unknown environments.