Monte Carlo and TD Learning

Anshul Choudhary

22 June, 2025

1 Introduction

In this task, I used six algorithms: **Monte Carlo, SARSA, Q-learning, Double-Q-Learning, Expected SARSA**, and **Policy Iteration** on two environments: **Frozen Lake** and **Cliff Walk**. We include a dynamic programming (DP) algorithm for comparison with model-free algorithms that do not assume complete knowledge of the environment. All five learning algorithms use ϵ**-greedy** policies.

2 Frozen Lake

2.1 10x10 grid

Algorithm	Time (s)	Avg Episode Length	Avg Reward	Episodes/Iterations
Monte Carlo	24.2642	63.633	0.937	40000
SARSA	1.0862	63.699	0.953	2000
Q-learning	1.6619	64.296	1.000	2000
Double Q-learning	1.6720	63.906	1.000	2000
Expected SARSA	1.9502	67.814	1.000	2000
Policy Iteration	0.0476	60.570	1.000	14

Table 1: Performance summary of RL algorithms on 10x10 grid

2.2 50x50 grid

Algorithm	Time (s)	Avg Epis. Len	Avg Reward	Episodes/Iterations
SARSA	163.4978	357.736	0.633	140000
Q-learning	119.0235	360.623	0.662	100000
Double Q-learning	161.8335	363.975	0.583	140000
Expected SARSA	109.1608	474.402	0.614	90000
Policy Iteration	1.9005	324.466	0.626	69

Table 2: Performance summary of RL algorithms on 50x50 grid

2.3 Training Method

Initially, I used constant epsilon and alpha values for training. This worked fine for a small environment like the 10x10 grid, but on the 50x50 grid, even after a million episodes, the

average reward was 0.

As the grid size increases, the probability that the agent will reach the goal by taking random actions becomes almost zero. This matters because, initially, all state-action values are zero, and since the only reward is at the goal, values remain zero.

One solution is to modify the reward structure, e.g., give a **-1 penalty for falling into a hole**, or increase rewards as the agent nears the goal. However, this could result in unintended behavior, such as the agent taking unnecessarily long routes.

To address this, I created two environments: one for **training** and one for **testing**. In the training environment, the agent starts at cell 0 with probability 0.1 and at other random cells with probability 0.9. This approximates **exploring starts** and ensures better value propagation. In the test environment the episode always started from cell 0 because that is the original problem we were trying to solve.

I also applied **epsilon and alpha decay** to accelerate convergence. Epsilon decay is essential in SARSA and Expected SARSA because the update depends on the next action, which should ideally be deterministic in later training. Alpha decay is also necessary because initially we should take big steps in updating the Q values but as the values approach the true Q values alpha should also approach 0 this is also a theoretical requirement for convergence: the sum of alpha should diverge, but the sum of its square should converge . I used linear decay for both learning rate(α) and epsilon.

2.4 10x10 grid result analysis

The environment used was a **10×10** custom Frozen Lake environment with **slipperiness enabled**.

All algorithms eventually learn the optimal policy. Any small variation in reward is due to the **stochastic nature** of the environment.

Policy Iteration is the fastest because it has full knowledge of the environment's dynamics. In contrast, the other algorithms must explore and learn by trial-and-error. **Monte Carlo** takes the most number of episodes and hence the highest time to converge to the optimal policy; the reason for this is that Monte Carlo only updates the action value functions after the end of an epsiode while the other algorithm learn during the episode too and thus are able to change and improve their policy during the episode itself.

2.5 50x50 grid result analysis

The environment used was a **50×50** custom Frozen Lake environment with **slipperiness enabled**. The observations are mostly consistent with the 10x10 case, except that even with improved training, **Monte Carlo fails to converge**.

Expected SARSA required fewer episodes to achieve a comparable reward but produced longer episodes.

3 Cliff Walking

Algorithm	Time (s)	Avg. Episode Length	Avg. Reward	Iterations
SARSA	0.642	17.0	-17.0	1000
Expected SARSA	0.806	17.0	-17.0	1000
Q-Learning	0.503	13.0	-13.0	1000
Double Q-Learning	0.418	13.0	-13.0	1000
Policy Iteration	0.007	13.0	-13.0	15

Table 3: Performance on Cliff Walking Environment

As seen in the book, **SARSA and Expected-SARSA learn the suboptimal but safe policy** by avoiding the cliff and taking a longer route. On the other hand, **Q-learning, Double Q-learning, and Policy Iteration** learn the optimal (shortest) policy.

In contrast to the book, Q-learning performs better than SARSA in my experiments. This is because the book tests with ϵ-greedy policies even during evaluation. Here, during testing, we use the **greedy policy without exploration**, allowing optimal algorithms to show their full potential. This doesn't mean Q-Learning is always preferable: SARSA learns safer policies at the cost of suboptimal reward. In real-world deployments, this may be necessary to avoid damage, whereas Q-learning may be better suited for simulation-based settings.