Tabular Methods for Frozen Lake MDP

Longping Chen
University of Illinois at Urbana-Champaign
lc36@illinois.edu

I. Introduction

This report mainly implements and compares three fundamental tabular methods, On-policy first-visit MC control, SARSA and Q-learning, for solving the Frozen Lake Markov Decision Process (MDP). [1] The Frozen Lake environment is a grid world problem which involves crossing a frozen lake from start to goal without falling into any holes by walking over the frozen lake.

A. MDP Formulation

The Frozen Lake MDP is formally defined as follows:
State Space (S): 16 discrete states arranged in a 4×4 grid:

0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15

Action Space (A): 4 discrete actions:

- 0: Left ($\leftarrow$)
- 1: Down ($\downarrow$)
- 2: Right ($\rightarrow$)
- 3: Up ($\uparrow$)

Transition Model ($P(s'|s,a)$): In our test, two configurations have been considered:

- Slippery: Transition mode is defined with slippery dynamics where each action has:
  - 1/3 probability of moving in the intended direction
  - 1/3 probability of moving perpendicular left
  - 1/3 probability of moving perpendicular right
- Non-slippery: The agent moves exactly in the intended direction

Reward Function ($R(s,a,s')$):

- $+1$ for reaching the goal state (G)
- 0 for all other transitions (including holes)

Discount Factor (γ): 0.95
Terminal States: Holes (H) and Goal (G) are terminal states with self-transitions.
Map Configuration: The map is configured as:

```
Row 0: "SFFF" → States 0,1,2,3
Row 1: "FHFH" → States 4,5,6,7
Row 2: "FFFH" → States 8,9,10,11
Row 3: "HFFG" → States 12,13,14,15
```

in which the terminal states are 5, 7, 11 (holes) and 15 (goal)

II. Method

A. Monte Carlo Control

Monte Carlo (MC) method learns value functions directly from sampled episodes without requiring a transition model. In first-visit MC control, each (s,a) pair is updated only upon its first occurrence in an episode. The return G_t following that visit is computed as:

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots$$

The action-value estimate is then updated as the sample average:

$$Q(s,a) \leftarrow Q(s,a) + \alpha\big[G_t - Q(s,a)\big]$$

In this implementation, an explicit learning rate α is not required since averaging is performed over all observed returns for (s,a). The policy improvement step follows an ε-greedy rule:

$$\pi(a|s) = \begin{cases} 1-\varepsilon+\dfrac{\varepsilon}{|A|}, & \text{if } a = \arg\max_{a'} Q(s,a') \\ \dfrac{\varepsilon}{|A|}, & \text{otherwise} \end{cases}$$

This balances exploration and exploitation, ensuring sufficient state-action coverage. The pseudocode of first-visit Monte Carlo Control is as shown in Algorithm 1.

Algorithm 1 First-Visit Monte Carlo Control

```
1: Initialize $Q(s,a)$ arbitrarily for all $s \in \mathcal{S}$, $a \in \mathcal{A}$
2: Initialize Returns$(s,a)$ as empty list for all $s,a$
3: Set $\varepsilon = \varepsilon_{\text{start}}$
4: for episode = 1 to $N$ do
5:     Generate episode following $\varepsilon$-greedy policy:
6:         $S_0, A_0, R_1, S_1, A_1, R_2, \dots, S_T$
7:     $G \leftarrow 0$
8:     for $t = T-1$ to 0 do
9:         $G \leftarrow \gamma \cdot G + R_{t+1}$
10:        if $(S_t, A_t)$ is first visit in episode then
11:            Append $G$ to Returns$(S_t, A_t)$
12:            $Q(S_t, A_t) \leftarrow$ average(Returns$(S_t, A_t)$)
13:        end if
14:    end for
15:    $\varepsilon \leftarrow \max(\varepsilon_{\text{end}}, \varepsilon \cdot \varepsilon_{\text{decay}})$
16: end for
```

B. SARSA: On-policy TD Control

SARSA (State–Action–Reward–State–Action) combines the advantages of dynamic programming and Monte Carlo methods. Unlike MC, SARSA performs online updates after every transition, thus reducing variance and enabling incremental learning.

The TD(0) update rule is given by:

$$Q(S,A) \leftarrow Q(S,A) + \alpha\Big[R + \gamma Q(S',A') - Q(S,A)\Big]$$

where (S', A') is the next state–action pair chosen under the same ε-greedy policy. This makes SARSA an on-policy algorithm, and the inclusion of α allows adaptive step sizes that trade off convergence speed and stability. The pseudocode of SARSA is as shown in Algorithm 2.

Algorithm 2 SARSA (On-policy TD Control)

```
 1: Initialize $Q(s,a)$ arbitrarily for all $s \in \mathcal{S}$, $a \in \mathcal{A}$
 2: Set $\alpha = 0.1$, $\varepsilon = \varepsilon_{\text{start}}$
 3: for episode $= 1$ to $N$ do
 4:     $S \leftarrow$ initial state
 5:     $A \leftarrow \varepsilon$-greedy$(Q, S, \varepsilon)$
 6:     while $S$ is not terminal do
 7:         Take action $A$, observe $R$, $S'$
 8:         $A' \leftarrow \varepsilon$-greedy$(Q, S', \varepsilon)$
 9:         $Q(S,A) \leftarrow Q(S,A) + \alpha[R + \gamma \cdot Q(S',A') - Q(S,A)]$
10:         $S \leftarrow S'$
11:         $A \leftarrow A'$
12:     end while
13:     $\varepsilon \leftarrow \max(\varepsilon_{\text{end}}, \varepsilon \cdot \varepsilon_{\text{decay}})$
14: end for
```

C. Q-learning: Off-policy TD Control

Q-learning is an off-policy TD control algorithm that learns the optimal policy independently of the agent's behavior. The update rule uses the greedy action in the next state, rather than the actual action taken:

$$Q(S,A) \leftarrow Q(S,A) + \alpha\Big[R + \gamma \max_a Q(S',a) - Q(S,A)\Big]$$

Here, the target value is based on the optimal estimate $\max_a Q(S', a)$, making Q-learning more aggressive and data-efficient. Even if the agent follows an ε-greedy exploration policy, Q-learning still converges to the optimal Q^* under suitable conditions. [2] The pseudocode of Q-Learning is as shown in Algorithm 3.

Algorithm 3 Q-Learning (Off-Policy TD Control)

```
 1: Initialize $Q(s,a)$ arbitrarily for all $s \in \mathcal{S}$, $a \in \mathcal{A}$
 2: Set $\alpha = 0.1$, $\varepsilon = \varepsilon_{\text{start}}$
 3: for episode $= 1$ to $N$ do
 4:     $S \leftarrow$ initial state
 5:     while $S$ is not terminal do
 6:         $A \leftarrow \varepsilon$-greedy$(Q, S, \varepsilon)$
 7:         Take action $A$, observe $R$, $S'$
 8:         $Q(S,A) \leftarrow Q(S,A) + \alpha[R + \gamma \cdot \max_a Q(S',a) - Q(S,A)]$
 9:         $S \leftarrow S'$
10:     end while
11:     $\varepsilon \leftarrow \max(\varepsilon_{\text{end}}, \varepsilon \cdot \varepsilon_{\text{decay}})$
12: end for
```

Across all three tabular reinforcement learning algorithms, standard hyperparameters were carefully selected to ensure a balance between learning stability, convergence speed, and computational efficiency. The discount factor was fixed at $\gamma = 0.95$, allowing future rewards to remain influential while maintaining preference for near-term gains. For exploration, an ε-greedy policy was employed, with the exploration rate initialized at $\varepsilon_0 = 1.0$ and exponentially decayed by a factor of 0.995 per episode until reaching a minimum of $\varepsilon_{\min} = 0.01$. For the TD algorithms (SARSA and Q-learning), the learning rate was set to $\alpha = 0.1$, which provided a stable trade-off between convergence speed and variance reduction. Each method was trained for 10,000 episodes to ensure sufficient state-action coverage and empirical convergence. These hyperparameter choices help to achieve consistent convergence behavior across both the slippery and non-slippery versions of the Frozen Lake environment.

III. Results

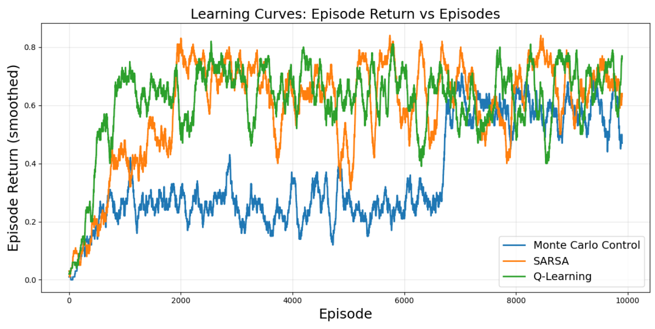
A. Convergence Analysis

Fig. 1a compares the convergence performance of the three model-free reinforcement learning algorithms for slippery scenario. As observed, all algorithms eventually achieve near-optimal behavior, though their convergence characteristics differ notably due to their respective learning paradigms.

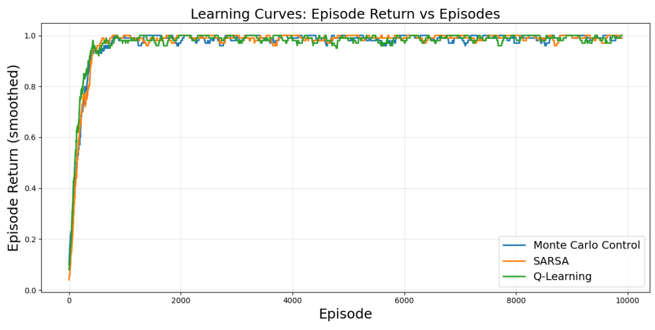
During the early training phase, which is around first 2000 episodes, SARSA and Q-learning exhibit a steep rise in average episode return, demonstrating the effectiveness of temporal-difference updates that enable faster propagation of reward information. In contrast, Monte Carlo Control progresses more gradually since it updates the value function only after full episodes are completed, resulting in higher variance and slower learning speed. Beyond approximately 4000 episodes, both SARSA and Q-learning stabilize around an average return of 0.7–0.8, indicating consistent success in reaching the goal state. Q-learning shows slightly faster convergence and higher asymptotic performance compared to SARSA, reflecting its off-policy nature that exploits the greedy target for value updates. However, SARSA maintains smoother learning curves, suggesting better stability under stochastic exploration. Monte Carlo Control eventually reaches a comparable performance level after about 6000.

Overall, the convergence behavior aligns with theoretical expectations: temporal-difference methods converge more efficiently than Monte Carlo approaches due to their bootstrapped updates. Among them, Q-learning achieves the best trade-off between learning speed and final performance, while SARSA provides greater robustness in noisy environments. These empirical results confirm that both on-policy and off-policy TD methods can effectively approximate the optimal value function in finite tabular MDPs.

For non-slippery scenario, the convergence performance of the three model-free reinforcement learning algorithms is as shown in Fig. 1b. The convergence of the three algorithms is basically the same, and they all complete training in about 500 episodes.



(a) Slippery scenario

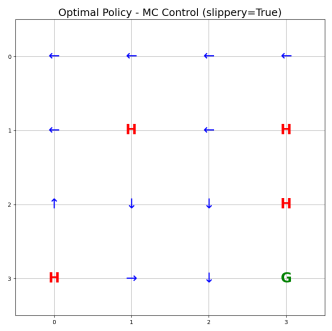


(b) Non-slippery scenario

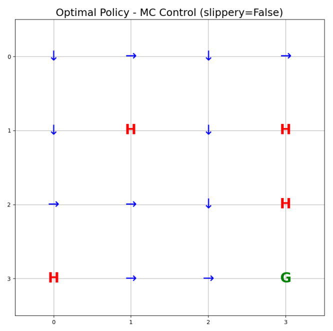
Fig. 1: Convergence behavior for slippery and non-slippery situation

B. Policy Comparison

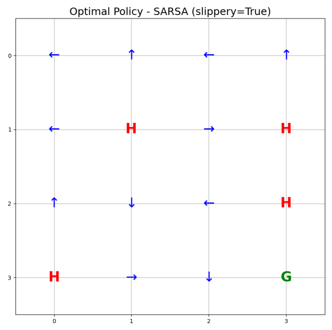
Fig. 2 shows the optimal policies obtained by the three algorithms, with arrows indicating the greedy action for each non-terminal state. In the slippery environment, as shown in Figures 2a, 2c, 2e, all algorithms demonstrate conservative navigation strategies, with SARSA exhibiting the most risk-averse behavior due to its on-policy nature, frequently choosing actions that maintain maximum distance from holes, while Q-Learning shows more aggressive goal-directed movements despite the stochastic transitions. Monte Carlo Control falls between these two extremes, learning cautious paths that avoid direct adjacency to dangerous states. In contrast, the non-slippery environment, as shown in Figures 2b, 2d, 2f, enables all algorithms to adopt more direct and efficient policies, with predominantly rightward and downward movements toward the goal; here, the differences between algorithms diminish significantly as the deterministic transitions reduce the need for conservative exploration. All three algorithms successfully learn viable hole-avoidance strategies, with states adjacent to hazards consistently showing actions that point away from danger, validating the effectiveness of model-free reinforcement learning in this gridworld domain.



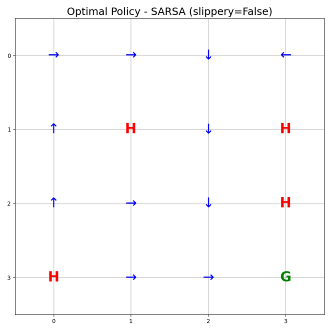
(a) MC Control (slippery)



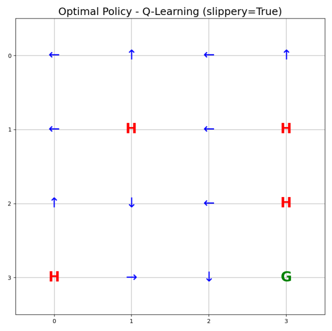
(b) MC Control (non-slippery)



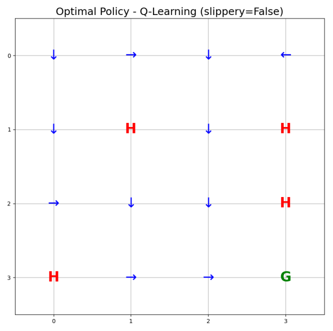
(c) SARSA (slippery)



(d) SARSA (non-slippery)



(e) Q-Learning (slippery)

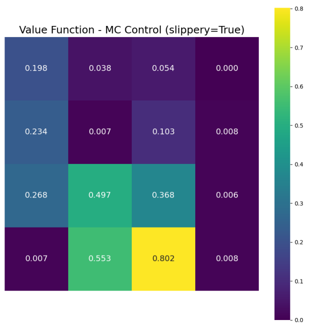


(f) Q-Learning (non-slippery)

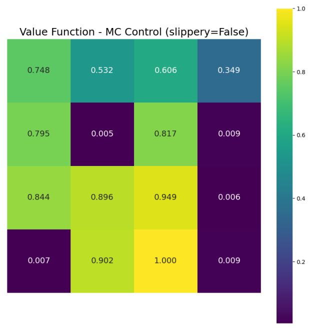
Fig. 2: Optimal policies for slippery and non-slippery situation

C. Value Function Comparison

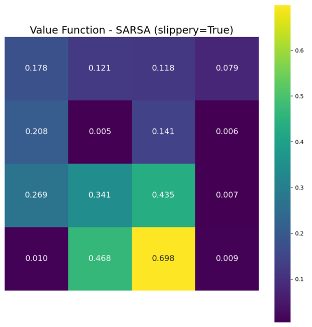
Fig. 3 shows the state value functions obtained by the three algorithms. The state value functions showed higher values near the goal state, with decreasing values as states are further from the goal. The value patterns reflect the optimal paths through the environment while accounting for the risk of falling into holes.



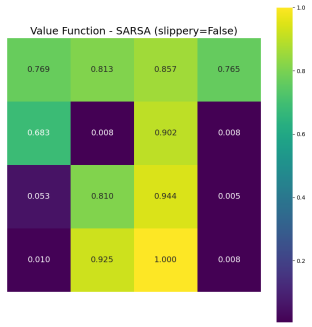
(a) MC Control (slippery)



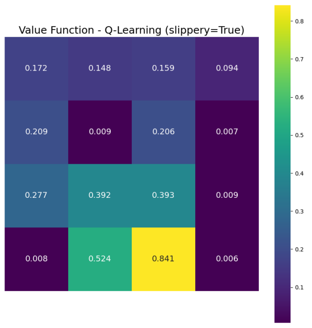
(b) MC Control (non-slippery)



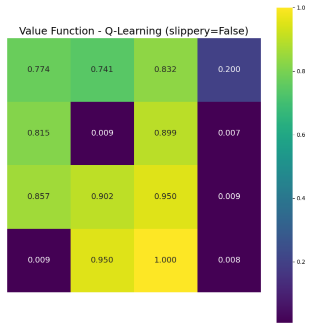
(c) SARSA (slippery)



(d) SARSA (non-slippery)



(e) Q-Learning (slippery)

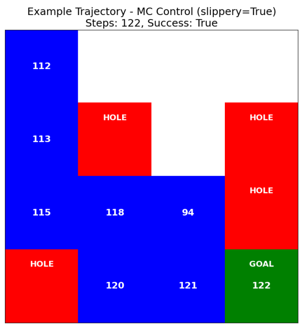


(f) Q-Learning (non-slippery)

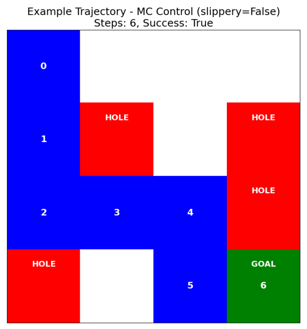
Fig. 3: State value functions for slippery and non-slippery situation

D. Example Trajectory Comparison

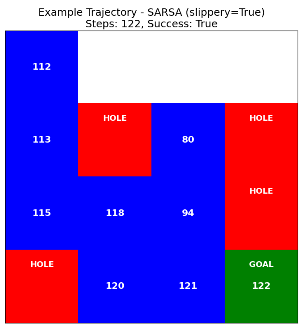
Fig. 4 shows the example trajectories obtained all three algorithms. In the figures, blue, green, red, and white squares represent trajectory paths, goals, holes, and walkable ice squares, respectively. In the slippery environment, as shown in Figures 4a, 4c, 4e, all three algorithms require significantly more steps to reach the goal due to stochastic transitions causing the agent to deviate from intended paths and revisit states multiple times, as evidenced by the extensive blue coverage throughout the grid. The trajectories show repeated oscillations and backtracking, with states being revisited numerous times before eventually reaching the goal at position. In contrast, the non-slippery environment, as shown in Figures 4b, 4d, 4f, enables all algorithms to achieve goal-reaching in just 6 steps via nearly optimal paths that move directly rightward and downward with minimal deviation, demonstrating the dramatic impact of environmental stochasticity on execution efficiency.


Example Trajectory - MC Control (slippery=True)
Steps: 122, Success: True
112
113
HOLE
HOLE
115
118
94
HOLE
HOLE
120
121
GOAL
122

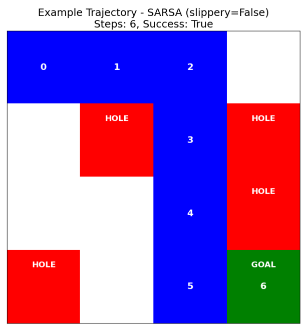
(a) MC Control (slippery)



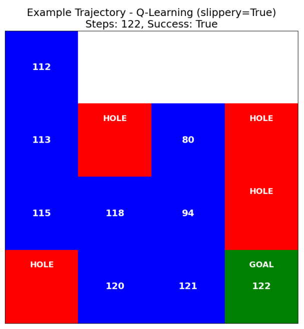
(b) MC Control (non-slippery)


Example Trajectory - SARSA (slippery=True)
Steps: 122, Success: True
112
113
HOLE
80
HOLE
115
118
94
HOLE
HOLE
120
121
GOAL
122

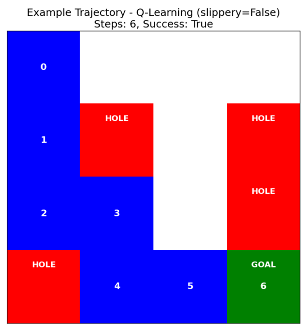
(c) SARSA (slippery)



(d) SARSA (non-slippery)


Example Trajectory - Q-Learning (slippery=True)
Steps: 122, Success: True
112
113
HOLE
80
HOLE
115
118
94
HOLE
HOLE
120
121
GOAL
122

(e) Q-Learning (slippery)



(f) Q-Learning (non-slippery)

Fig. 4: Example trajectories for slippery and non-slippery situation

IV. Conclusions

This report compared the performance of three fundamental tabular methods for solving the Frozen Lake MDP under both slippery and non-slippery scenarios. The experimental results demonstrate that all three algorithms can effectively learn viable policies for navigating the gridworld environment, but they exhibit distinct convergence characteristics and behavioral patterns. Specifically, Q-Learning consistently achieved the fastest convergence and highest final performance due to its off-policy nature, which allows aggressive optimization by learning from greedy target values independent of exploration behavior. SARSA, as an on-policy method, demonstrated more conservative and stable learning, particularly in the stochastic slippery environment where it maintained greater safety margins from holes. Monte Carlo Control, while requiring more episodes to converge due to its reliance on complete episode returns, eventually reached comparable performance levels with higher variance throughout training.

The comparison between slippery and non-slippery environments revealed that environmental stochasticity fundamentally impacts both learning efficiency and policy characteristics. In deterministic settings, all algorithms converged rapidly within approximately 500 episodes and learned direct goal-oriented policies, whereas the stochastic environment required significantly longer training and induced risk-averse navigation strategies. The trajectory analysis further confirmed this impact, with slippery environments requiring approximately 20 times more steps to reach the goal compared to deterministic settings.

References

[1] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. Cambridge: MIT press, 2018.

[2] C. J. Watkins and P. Dayan, “Q-learning,” *Machine learning*, vol. 8, no. 3, pp. 279–292, 1992.