# Reinforcement Learning: Assignment 3
# DSCI-6650

Soroush Baghernezhad
sbaghernezha@mun.ca

August 7, 2024

**Abstract**

In this assignment, we interact with a grid world and try different Temporal Difference learning approaches and function approximation methods to achieve the optimal policy in the environment and predict the value estimation, respectively. We also compare different methods to see how they perform and what is their final policy.

## 1 Introduction

The environment is a 5*5 grid with two terminal states on top, in (0,0) and (0,4), four red tiles corresponding to a wall which if the agent step in, it will get a -20 reward and go back to the start. The red tiles are located at (2,0),(2,1),(2,3),(2,4).
The starting state is on the (4,0) indicated with a blue color. The goal is passing through the center of the red wall and getting to the one of the terminal states. Each step on empty states will cause a reward of -1.
in this environment and this part, our objective is using different temporal difference learning methods and comparing them with each other.
In the second part, there is a 7*7 grid, we need to perform a random walk on it with the equal probability of goind up, down, left or right. We start at the center of the grid and there are two terminal states at the upper right corner with +1 reward and lower left corner with -1 reward. All other movements give 0 reward and the goal is getting to the terminal states.
Our objective in this part is using function approximation to estimate the value function of the random walk.

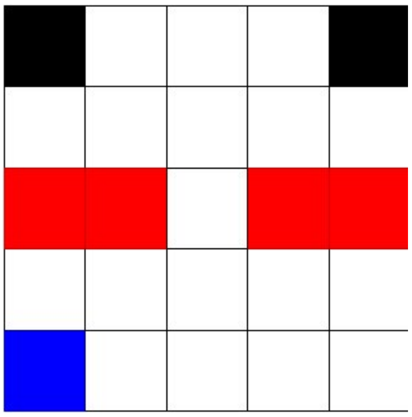
Figure 1: grid for first part

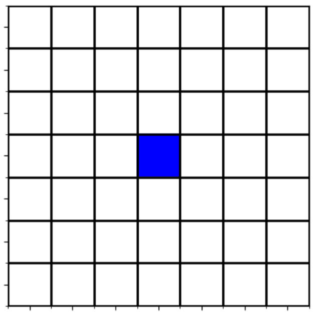
Figure 2: grid for second part

## 2 Part One

In this part, we implemented four different TD approaches, namely, Sarsa, Q-learning, Expected Sarsa and Double Q-learning. In the following section we will briefly discuss each method.

### 2.1 Sarsa: On-policy TD control

TD prediction methods are employed for control problems using Generalized Policy Iteration (GPI). The Sarsa algorithm is an on-policy TD control algorithm, which means it estimates the action-values based on the current behavior policy $\pi$ and for all states $s$ and actions $a$. This involves learning the values of state-action pairs rather than just states, resembling Markov chains with a reward process. The Sarsa algorithm, an on-policy TD control method, updates $Q(s, a)$ after every transition from a non-terminal state. If the next state is terminal, $Q(s_t + 1, a_t + 1)$ is set to zero. Sarsa's updates are based on sequences of state-action-reward-state-action (Sarsa) transitions. The algorithm iteratively estimates $q_\pi$ for the behavior policy while steering $\pi$ towards greediness relative to $q_\pi$.

Below you can find Sarsa algorithm:

**Algorithm 1** Sarsa Algorithm

```
0: Algorithm parameters: step size $\alpha \in (0, 1]$, small $\epsilon > 0$
0: Initialize $Q(s, a)$ for all $s \in S^+$, $a \in A(s)$, arbitrarily except that $Q(\text{terminal}, \cdot) = 0$
0: while True do {Loop for each episode}
0:    Initialize $S$
0:    Choose $A$ from $S$ using policy derived from $Q$ (e.g., $\epsilon$-greedy)
0:    while $S$ is not terminal do {Loop for each step of episode}
0:       Take action $A$, observe $R$, $S'$
0:       Choose $A'$ from $S'$ using policy derived from $Q$ (e.g., $\epsilon$-greedy)
0:       $Q(S, A) \leftarrow Q(S, A) + \alpha[R + \gamma Q(S', A') - Q(S, A)]$
0:       $S \leftarrow S'$
0:       $A \leftarrow A'$
0:    end while
0: end while=0
```

### 2.2 Q-learning: Off-policy TD Control

Q-learning is one of the early breakthroughs in reinforcement learning, In contrast with Sarsa, in Q-learning, the learned action-value function, Q, directly approximates $q_*$, the optimal action-value function, independent of the policy being followed.

**Algorithm 2** Q-Learning Algorithm

```
0: Algorithm parameters: step size $\alpha \in (0, 1]$, small $\epsilon > 0$
0: Initialize $Q(s, a)$ for all $s \in S^+$, $a \in A(s)$, arbitrarily except that $Q(\text{terminal}, \cdot) = 0$
0: while True do {Loop for each episode}
0:   Initialize $S$
0:   while $S$ is not terminal do {Loop for each step of episode}
0:     Choose $A$ from $S$ using policy derived from $Q$ (e.g., $\epsilon$-greedy)
0:     Take action $A$, observe $R$, $S'$
0:     $Q(S, A) \leftarrow Q(S, A) + \alpha[R + \gamma \max_a Q(S', a) - Q(S, A)]$
0:     $S \leftarrow S'$
0:   end while
0: end while=0
```

## 2.3 Expected Sarsa:

This algorithm is similar to Q-learning except instead of taking maximum over the next state-action pairs, it uses the expected values. Although this method is computationally more expensive than sarsa, it eliminates the variance due to the random selection of $A_t+1$ and it performs slightly better than it.
The pseudo code of Expected Sarsa is as follow:

**Algorithm 3** Expected Sarsa Algorithm

```
0: Algorithm parameters: step size $\alpha \in (0, 1]$, small $\epsilon > 0$
0: Initialize $Q(s, a)$ for all $s \in S^+$, $a \in A(s)$, arbitrarily except that $Q(\text{terminal}, \cdot) = 0$
0: while True do {Loop for each episode}
0:   Initialize $S$
0:   while $S$ is not terminal do {Loop for each step of episode}
0:     Choose $A$ from $S$ using policy derived from $Q$ (e.g., $\epsilon$-greedy)
0:     Take action $A$, observe $R$, $S'$
0:     Compute expected value: $\mathbb{E}[Q(S', a)] = \sum_a \pi(a|S')Q(S', a)$
0:     $Q(S, A) \leftarrow Q(S, A) + \alpha[R + \gamma\mathbb{E}[Q(S', a)] - Q(S, A)]$
0:     $S \leftarrow S'$
0:   end while
0: end while=0
```

$$\text{where} \quad \mathbb{E}_\pi[Q(S_{t+1}, A_{t+1}) \mid S_{t+1}] = \sum_a \pi(a \mid S_{t+1})Q(S_{t+1}, a)$$

## 2.4 Double Q-learning

Double Q-Learning is an enhancement to the standard Q-Learning algorithm designed to address some of its key limitations, particularly overestimation bias. In standard Q-Learning, the Q-value update rule uses the maximum estimated Q-value of the next state to compute the target value. This can lead to overestimation of action values because the same Q-values are used both to select and to evaluate actions.

Double Q-Learning mitigates this issue by using two separate Q-value functions. These two Q-value functions are updated independently, which helps to reduce overestimation bias.

**Algorithm 4** Double Q-Learning Algorithm

```
0: Algorithm parameters: step size $\alpha \in (0, 1]$, small $\epsilon > 0$
0: Initialize $Q_1(s, a)$ and $Q_2(s, a)$ for all $s \in S^+$, $a \in A(s)$, arbitrarily except that
   $Q_1(\text{terminal}, \cdot) = 0$ and $Q_2(\text{terminal}, \cdot) = 0$
0: while True do {Loop for each episode}
0:    Initialize $S$
0:    while $S$ is not terminal do {Loop for each step of episode}
0:       Choose $A$ from $S$ using $\epsilon$-greedy policy derived from $Q_1 + Q_2$
0:       Take action $A$, observe $R$, $S'$
0:       With probability 0.5:
0:          $Q_1(S, A) \leftarrow Q_1(S, A) + \alpha [R + \gamma Q_2(S', \arg\max_a Q_1(S', a)) - Q_1(S, A)]$
0:       Else:
0:          $Q_2(S, A) \leftarrow Q_2(S, A) + \alpha [R + \gamma Q_1(S', \arg\max_a Q_2(S', a)) - Q_2(S, A)]$
0:       $S \leftarrow S'$
0:    end while
0: end while=0
```

## 2.5 Experiment Setup

We used similar sets of parameters for each algorithm:

- $n - episodes$: the number of episodes
- $\alpha$: learning rate
- $\epsilon$: epsilon used in epsilon-greedy policy
- $\gamma$: discount factor
- $\epsilon_{decay}$: decay rate of epsilon through the learning

As the book suggested, we set $\alpha$ to 0.1, $\gamma$ to $discount = 0.95$ and $\epsilon_{decay}$ to 0.00005, other parameters vary for convergence.

We use epsilon decay parameter to reduce the exploration after each episode, this helps our algorithm to converge faster and not stuck in a loop or return to the other side of the wall after passing the red wall.
We also tried different parameters in parameter tuning part to see the effect of each parameter on the convergence and the policy learnt by each algorithm.

## 3 Results: Part One

We first use 500 episodes and run all four algorithms to see if any of these algorithms can converge to an optimal policy in these few episodes or not. We get our results from four experiments each with a different episode number, respectively, 500, 1000, 5000 and 10000.
For better comparing the similarity and differences of algorithms, we plotted three different charts:

- Policy Grid: in which we show the policy derived by maximizing over the $Q$ of the last episode of each algorithm.
- Steps per episodes: in which we show the average time steps took for an algorithm to find the terminal state during different episodes.
- sum of rewards per episodes: total reward accumulated over different episodes.

Initially we start running Sarsa with different sets of parameters (e.g. $\epsilon = .05 \;\&\; \alpha = 0.5$) but it diverged . Then we used other parameters that were suggested by the book and they are:
$\alpha = 0.1$ $\epsilon = 0.2$ $\gamma = 0.95$ $\epsilon_{decay} = 0.00005$
and eventually it started converging.

As it can be seen in Figure 3, none of the methods could find a policy that can reach to terminal state, however, Sarsa and Expected Sarsa seem to find out about how to pass the red wall. The whole experiments took less than 1 minute.
Figure 4 shows that there is a sharp fall after first 20 episodes in the average time steps that took for agent to find the terminal states which shows that algorithms are learning to find the terminal states faster and better.

Figure 5 depicts the average sum of rewards over episodes, and all algorithms are trying to get better sum of rewards. Note that sum of rewards cannot be zero and the best possible sum of reward that can be obtained is equal to the shortest distance from starting point to one of terminal states which is -8.

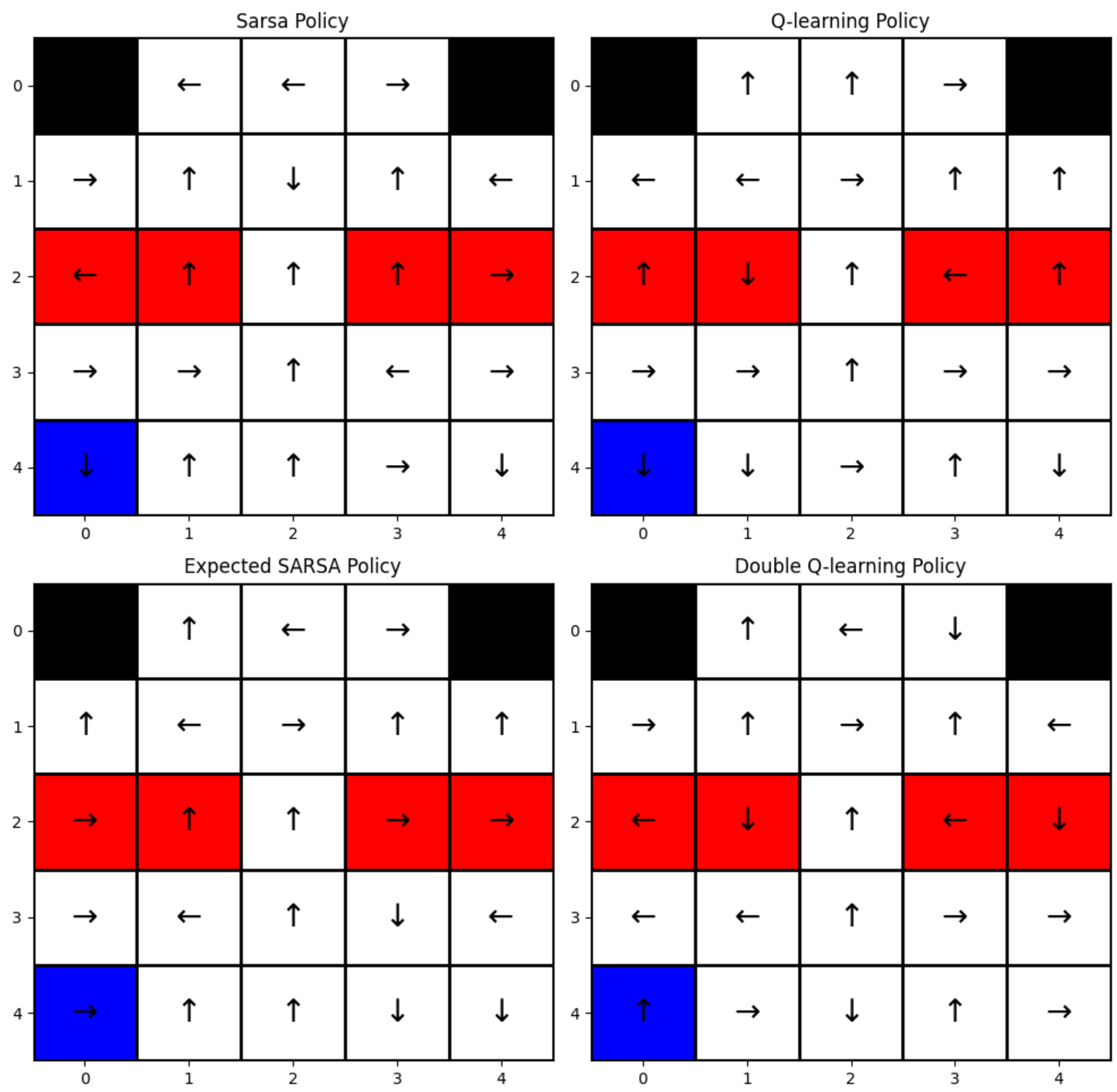

Figure 3: First Experiment: Policy of all four algorithms after 500 episodes.

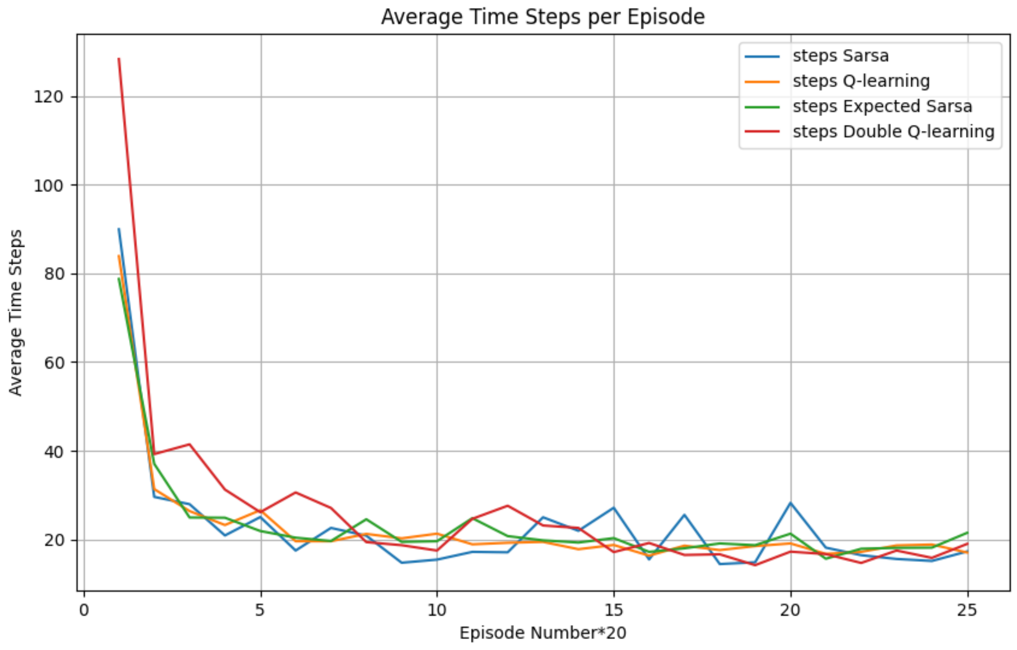

Figure 4: First Experiment: Average time steps to complete an episode over number of episodes for all four algorithms.

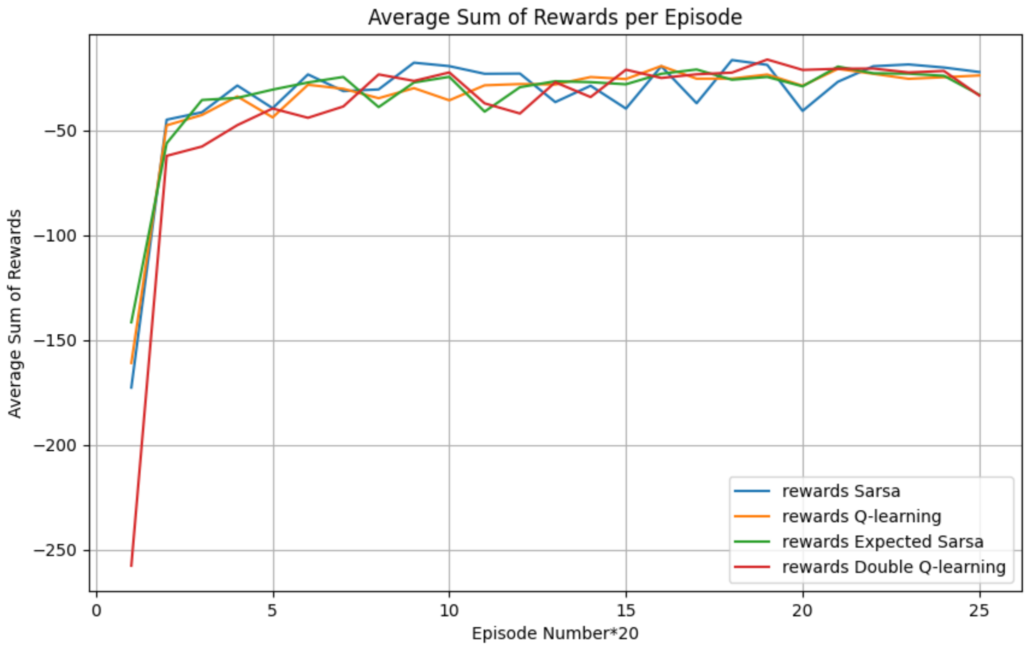

Figure 5: First Experiment: Average sum of rewards per episodes for all four algorithms.

For the second experiment, we chose 1000 episodes to run all four algorithms on and same set of parameters as first experiment:

$\alpha = 0.1 \ \epsilon = 0.2 \ \gamma = 0.95 \ \epsilon_{decay} = 0.00005$

In Figure 6, only Sarsa could find a policy that reach the terminal state in 1000 episodes, Expected Sarsa is close too but Q-learning approaches needs more episodes to get to a good policy.

Figure 7 shows the variance that Sarsa algorithm have during the training due to its stochastic selection of subsequent actions.

Figure 8 shows the correlation between number of steps and the sum of rewards, as sarsa spikes in Figure 7, there is a huge negative reward on the same episode. This observation is trivial because as agent explores more in the grid, it gets more negative rewards.

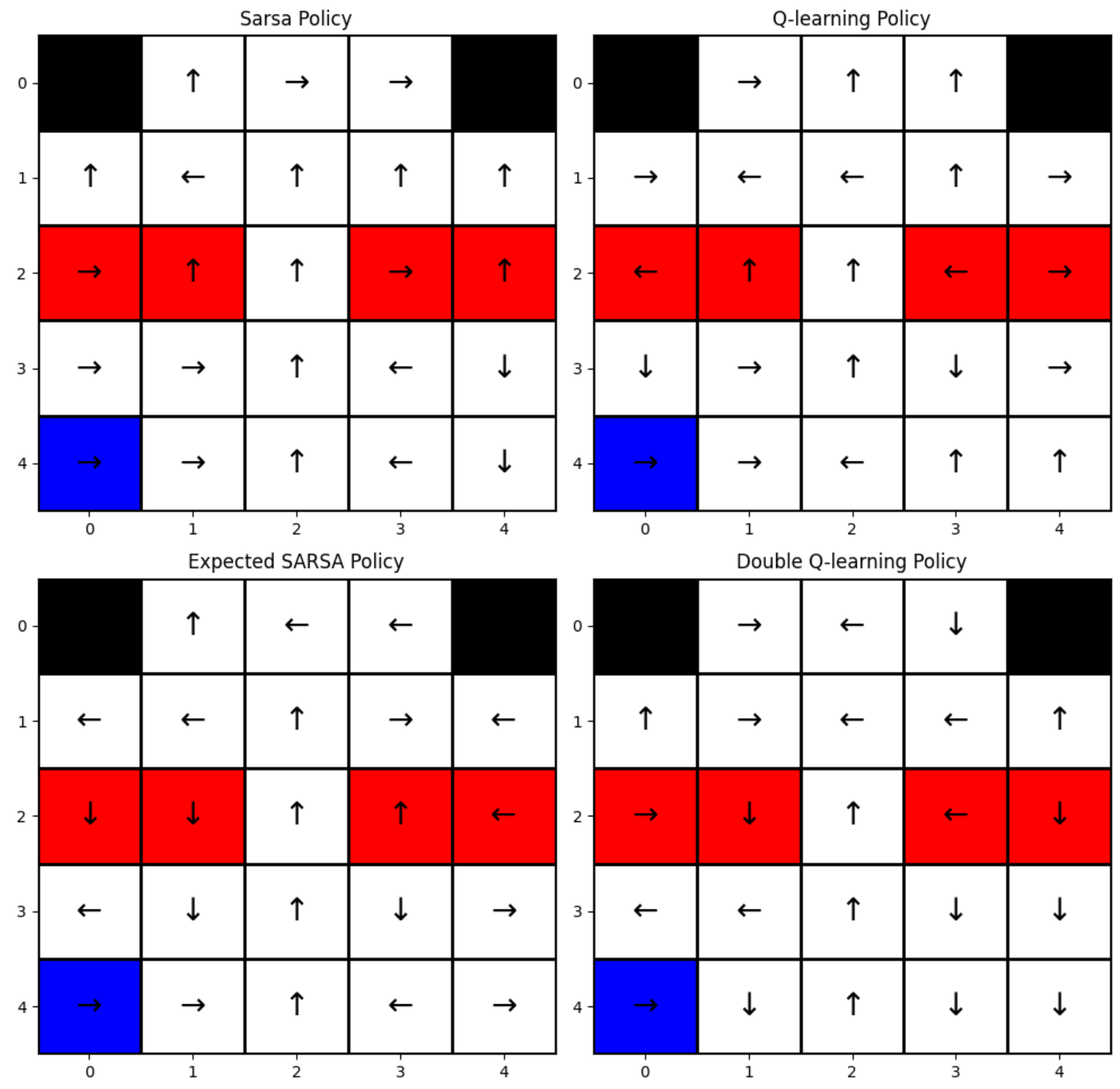

Figure 6: Second Experiment: Policy of all four algorithms after 1000 episodes.

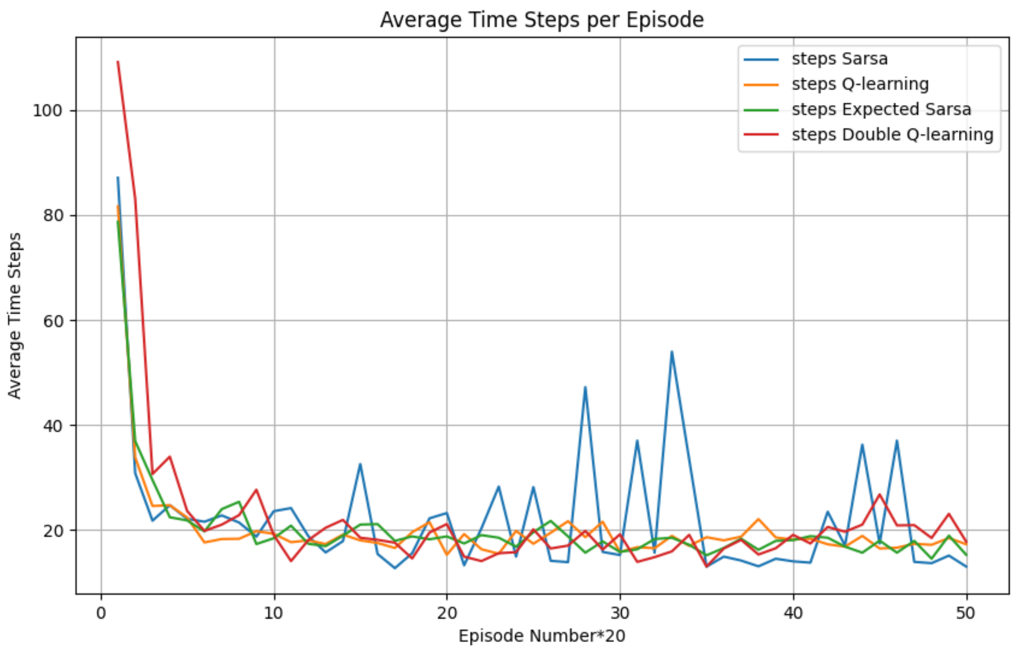

Figure 7: Second Experiment: Average time steps to complete an episode over number of episodes for all four algorithms.

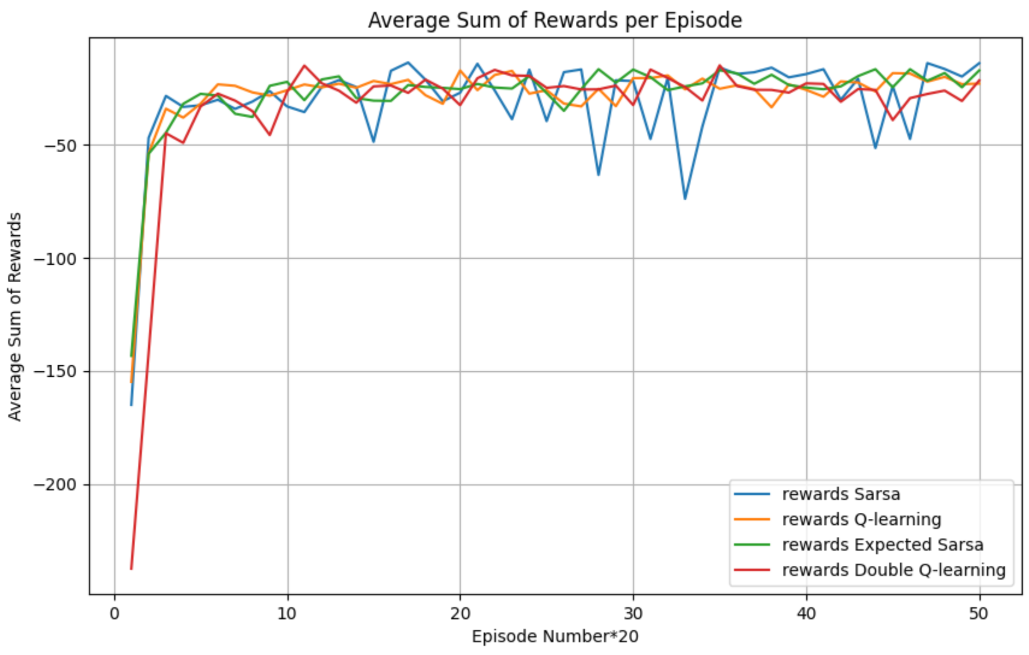

Figure 8: Second Experiment: Average sum of rewards per episodes for all four algorithms.

In third experiment we increased the number of episodes to 5000 and the final policies

derived from Q are shown in Figure 9.
Very similar to Cliff Walking of the book, Q-learning algorithm tend to choose a path near to wall with huge negative rewards where as Sarsa algorithm choose a safer path to the terminal state.

Figure 10 shows the huge variance of Sarsa algorithm, and in Figure 11 we can clearly see that sum of rewards obtained by Double Q-learning and Sarsa is less (better) than two others and they converged to the optimal value whereas Expected Sarsa and Q-learning converged to a non-optimal point.

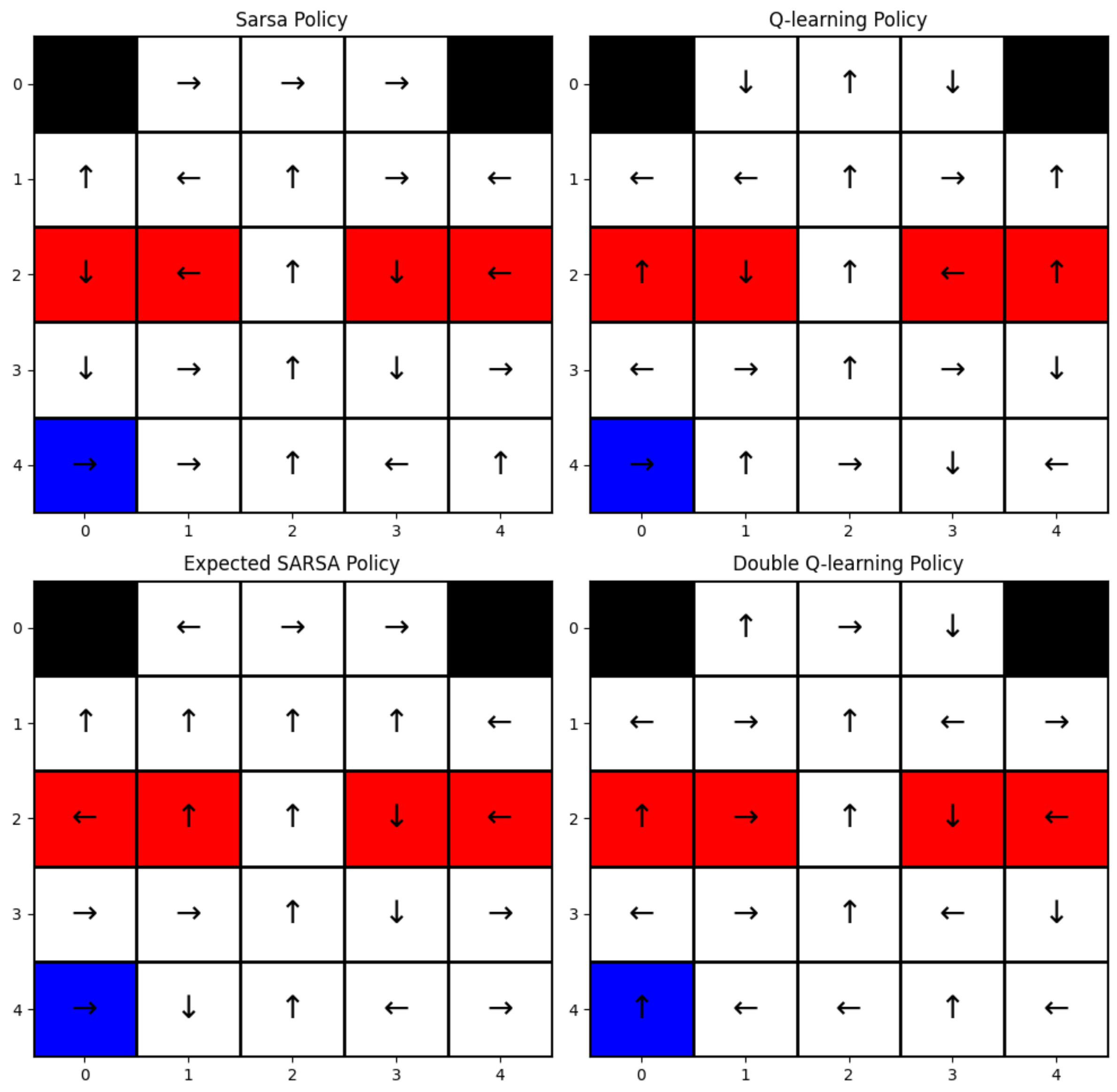

Figure 9: Third Experiment: Policy of all four algorithms after 5000 episodes.

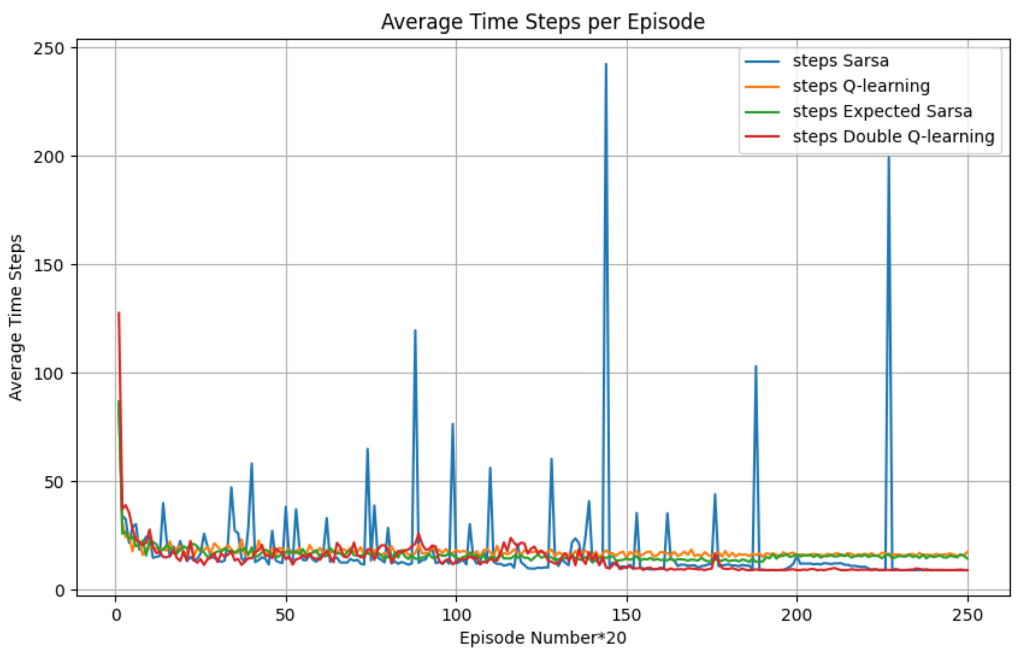

Figure 10: Third Experiment: Average time steps to complete an episode over number of episodes for all four algorithms.

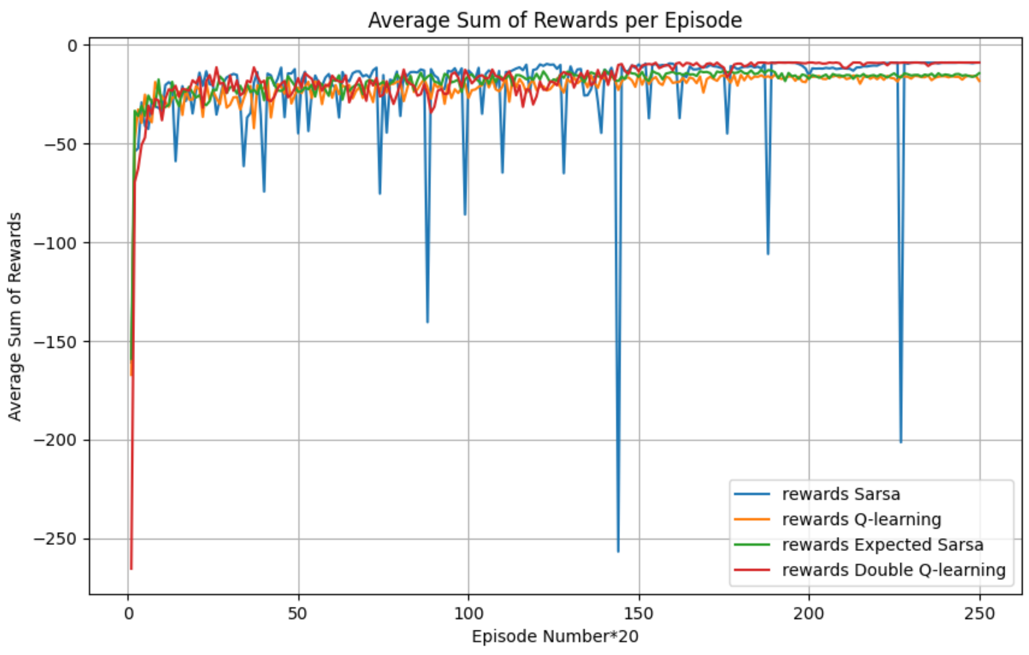

Figure 11: Third Experiment: Average sum of rewards per episodes for all four algorithms.

For the fourth experiment, we set the number of episodes to 10000 and for the few couple of initial runs, Sarsa and Expected Sarsa did not converge, for example, it took 34 minutes for expected Sarsa to finish 8000 episodes and 25 minutes to reach 8400 episodes from 8000, this is a sign of divergence in the algorithm, hence we stopped the restart the experiment to finally getting following results.

After running algorithms for 10000 episodes, none of them could find an optimal policy Figure 12.
Figure 13 and 14 shows that Sarsa struggled to find the terminal states and it spend 40,000 or even more steps to finish a single episode. Perhaps different set of parameters should be taken to find an optimal policy in large number of episodes.

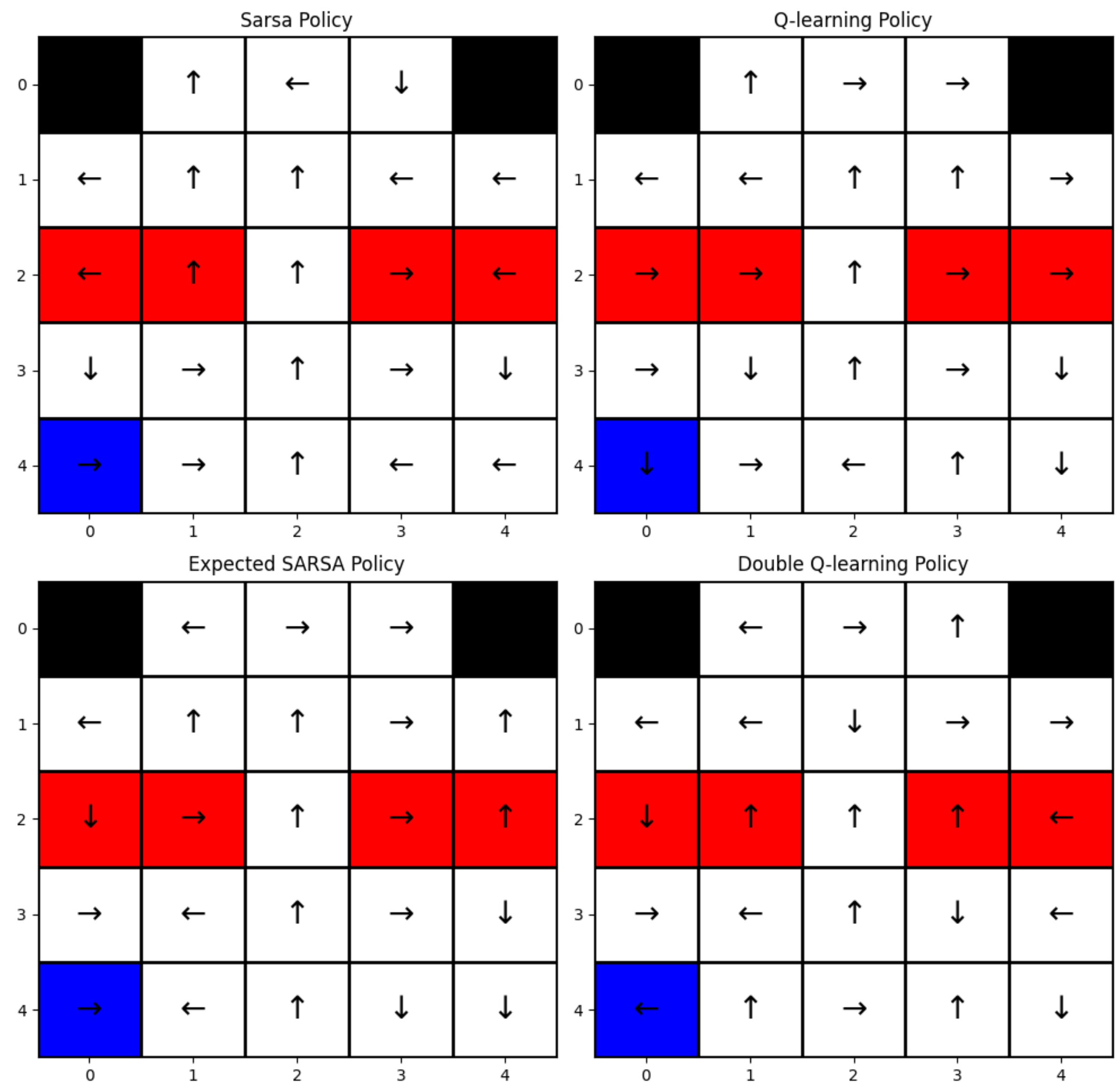

Figure 12: Fourth Experiment: Policy of all four algorithms after 10000 episodes.

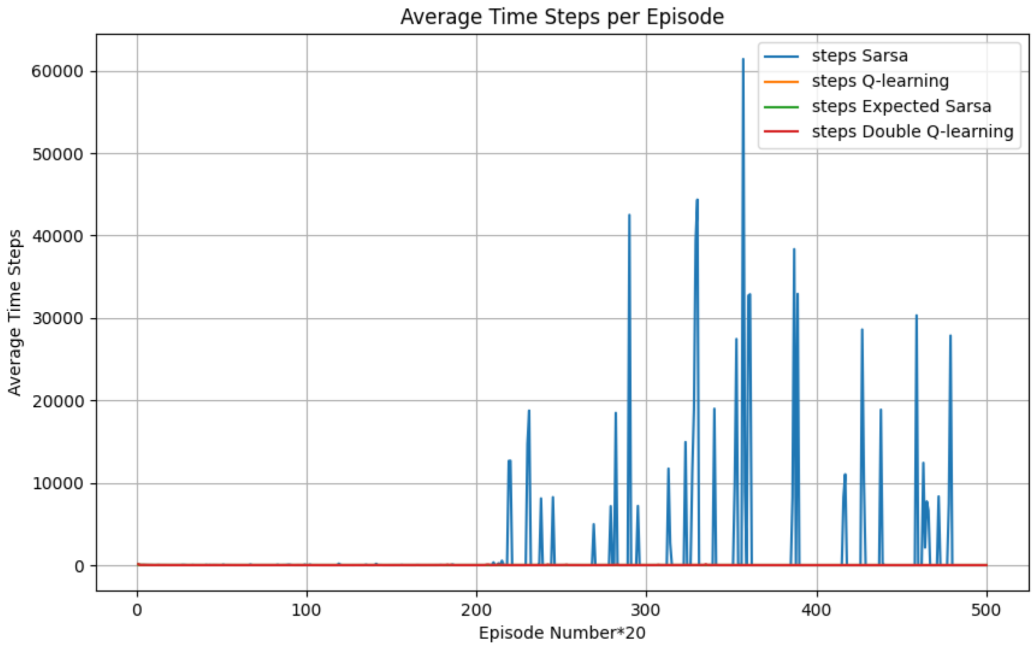

Figure 13: Fourth Experiment: Average time steps to complete an episode over number of episodes for all four algorithms.

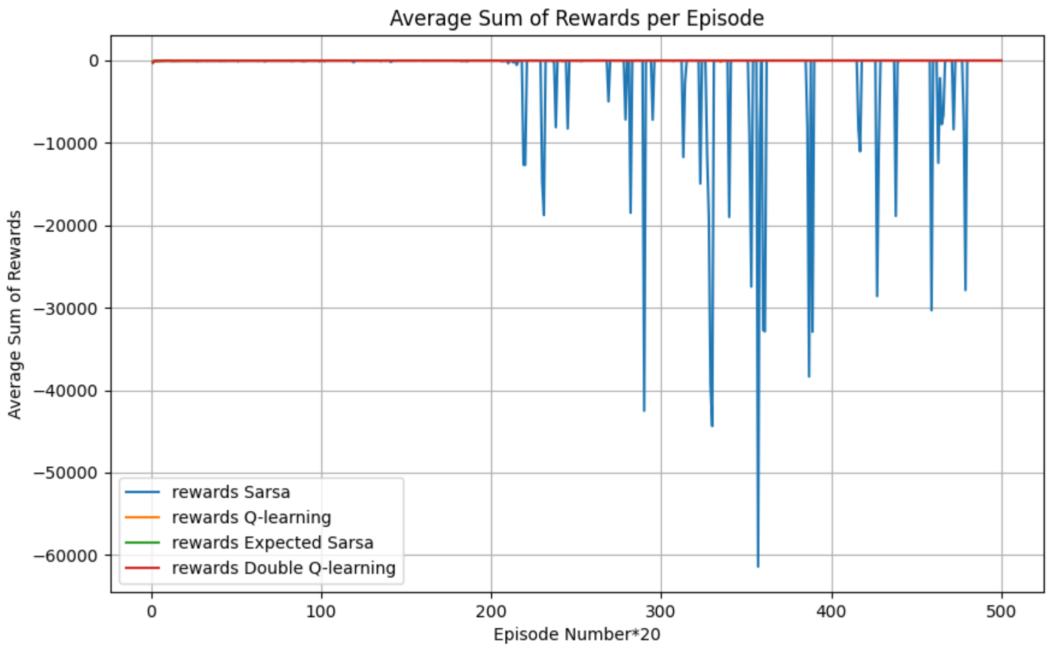

Figure 14: Fourth Experiment: Average sum of rewards per episodes for all four algorithms.

## 4 Part Two

In this part, we used a modified version of the grid in which two terminal states with black color are present in (4,0) and (2,4) positions. This terminal states will end an episode and return the reward accumulated. For estimating the value function of this grid, we used Gradient Monte Carlo and Semi Gradient TD which uses function approximation. Function approximation involves using parameterized functions, such as neural networks or linear combinations of features, to estimate value functions or policies. This allows algorithms to generalize from limited data to unseen states or actions.

In this part, we used one-hot encoded vector of states as the feature states of our two methods. We also solved bellman equation to find the true values of states in the grid.

### 4.1 Gradient Monte Carlo:

Here you can see the Gradient Monte Carlo pseudo code:

**Algorithm 5** Gradient Monte Carlo

```
0: Input: the policy $\pi$ to be evaluated
0: Input: a differentiable function $\hat{v} : S \times \mathbb{R}^d \to \mathbb{R}$
0: Algorithm parameter: step size $\alpha > 0$
0: Initialize: value-function weights $w \in \mathbb{R}^d$ arbitrarily (e.g., $w = 0$)
0: while True do {forever (for each episode)}
0:    Generate an episode $S_0, A_0, R_1, S_1, A_1, \ldots, R_T, S_T$ using $\pi$
0:    for each step of episode, $t = 0, 1, \ldots, T-1$ do
0:       $G_t \leftarrow \sum_{k=t+1}^{T} \gamma^{k-t-1} R_k$
0:       $w \leftarrow w + \alpha(G_t - \hat{v}(S_t, w))\nabla_w \hat{v}(S_t, w)$
0:    end for
0: end while=0
```

### 4.2 Semi Gradient TD(0)

This is pseudo code of the Semi Gradient TD:

**Algorithm 6** Semi-Gradient TD(0)

```
0: **Input:** the policy $\pi$ to be evaluated
0: **Input:** a differentiable function $\hat{v} : S^+ \times \mathbb{R}^d \to \mathbb{R}$ such that $\hat{v}(\text{terminal}, \cdot) = 0$
0: **Algorithm parameter:** step size $\alpha > 0$
0: **Initialize:** value-function weights $w \in \mathbb{R}^d$ arbitrarily (e.g., $w = 0$)
0: **for** each episode **do**
0:   Initialize $S$
0:   **while** $S$ is not terminal **do**
0:     Choose $A \sim \pi(\cdot|S)$
0:     Take action $A$, observe $R$, $S'$
0:     $w \leftarrow w + \alpha\,[R + \gamma\hat{v}(S', w) - \hat{v}(S, w)]\,\nabla_w \hat{v}(S, w)$
0:     $S \leftarrow S'$
0:   **end while**
0: **end for**=0
```

## 5 Result: Part Two

For comparing results, we plotted the value function of each algorithm beside the true values obtained by solving bellman equation. Additionally, we plotted the value scale per states as page 204 of the book.

After running the experiments for different set of episodes (1k, 5k, 10k, 20k, 50k), We found out that running for 10,000 episodes will give the best results, hence, we followed our experiments with fixed number of episodes but varying learning rate of $0.1, 0.2, 0.3, 0.5$.

In Figure 15 it can be seen that Semi Gradient TD could efficiently converge to the true values and the transition of values between two corners are as smooth as true values grid, on the other hand, Gradient MC could not get close to the true values and state values are not updated like the true values. This difference is due to the randomness of action selection and some states may not be visited during the episodes.

Figure 16 clearly shows how close Semi GRadient TD and Bellman equations are.
In Figures 18 20 22 we can see that with increasing the learning rate, both algorithms fail to get close to the true values, for example, with learning rate of 0.5, Semi Gradient TD surpass the true values of states.

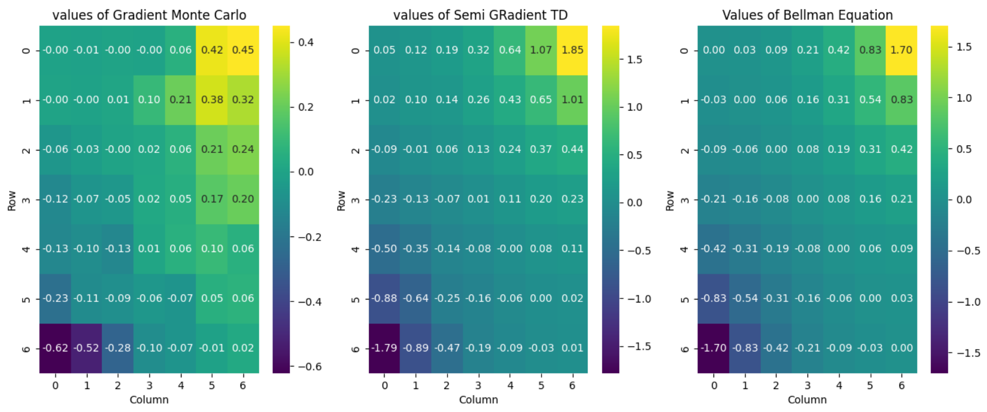

Figure 15: First Experiment: values after 10000 episodes $\alpha = 0.1$

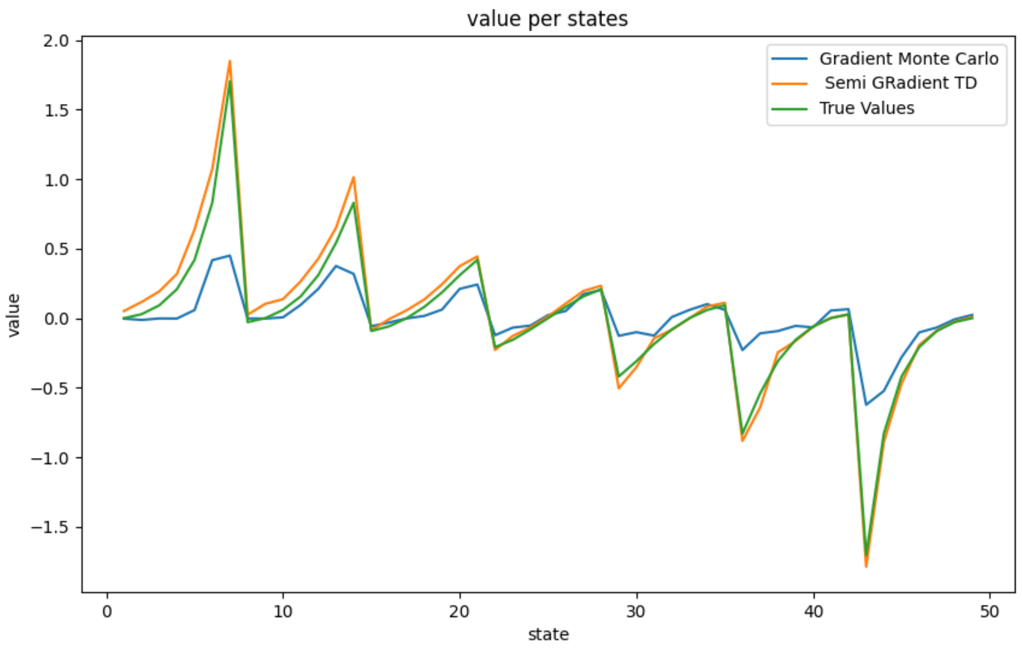

Figure 16: First Experiment: Value of each state $\alpha = 0.1$

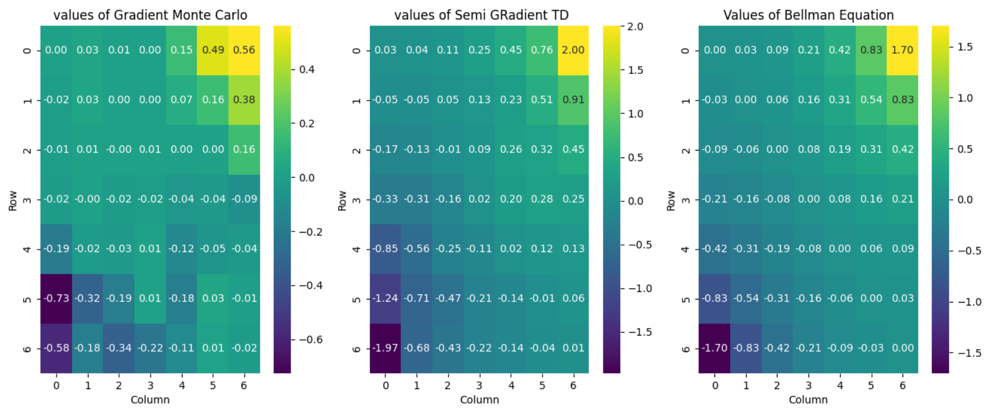

Figure 17: First Experiment: values after 10000 episodes $\alpha = 0.2$

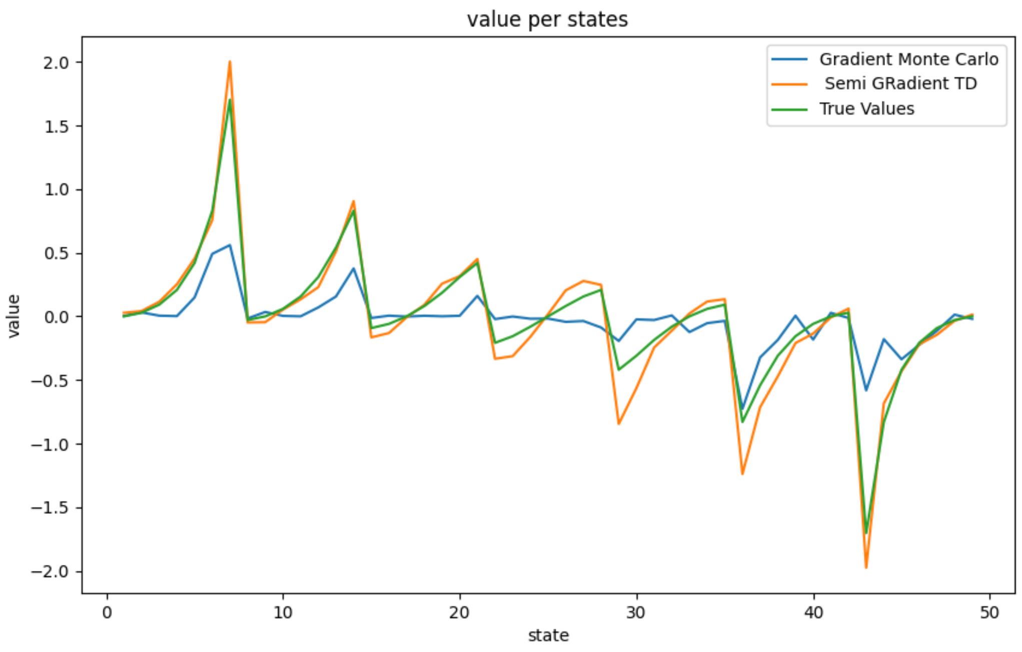

Figure 18: First Experiment: Value of each state $\alpha = 0.2$

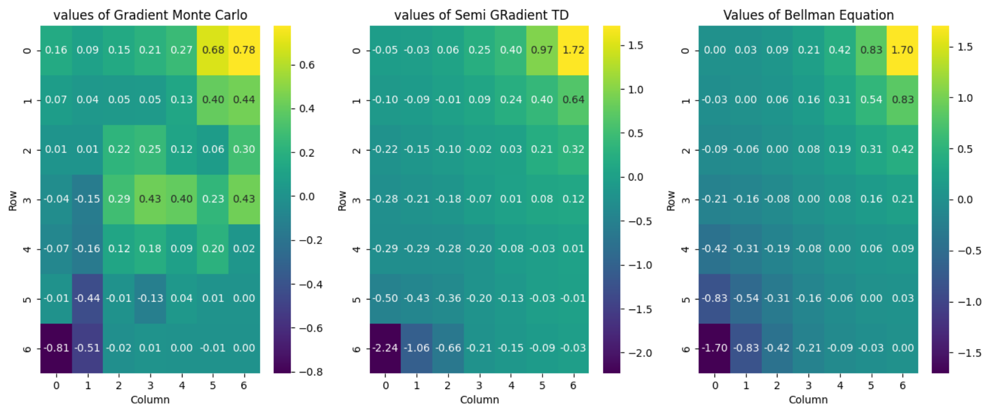

Figure 19: First Experiment: values after 10000 episodes $\alpha = 0.3$

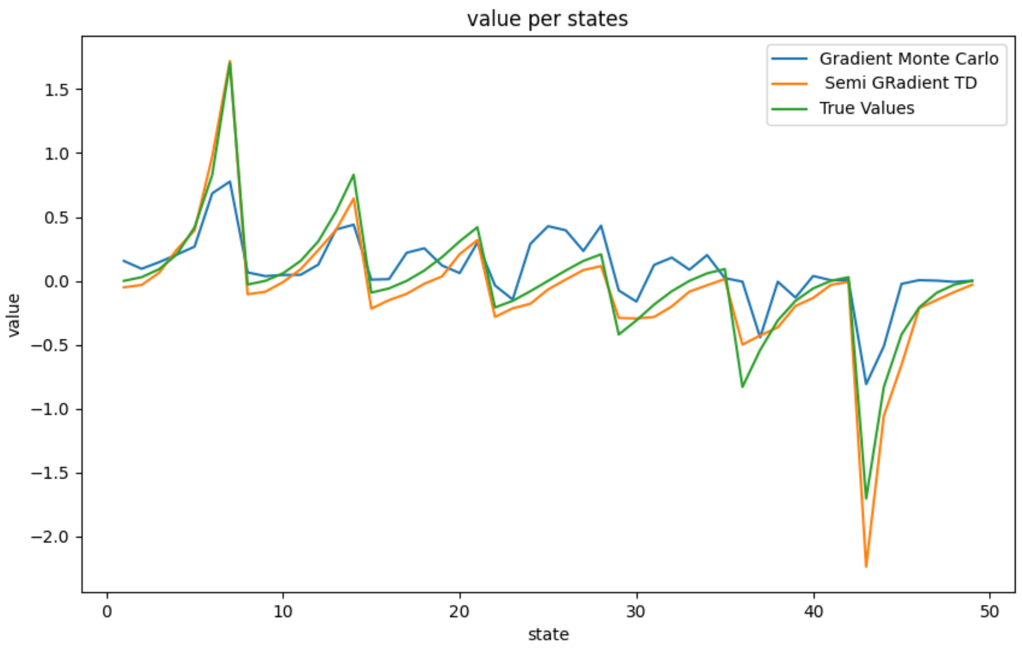

Figure 20: First Experiment: Value of each state with $\alpha = 0.3$

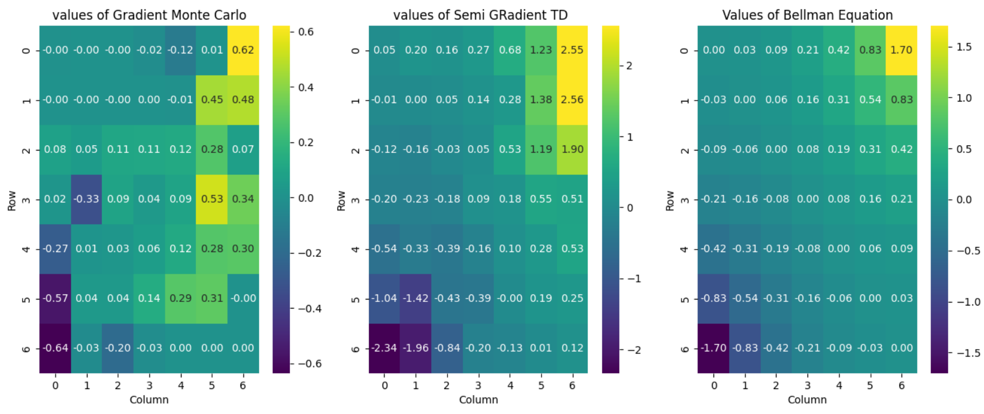

Figure 21: First Experiment: values after 10000 episodes $\alpha = 0.5$

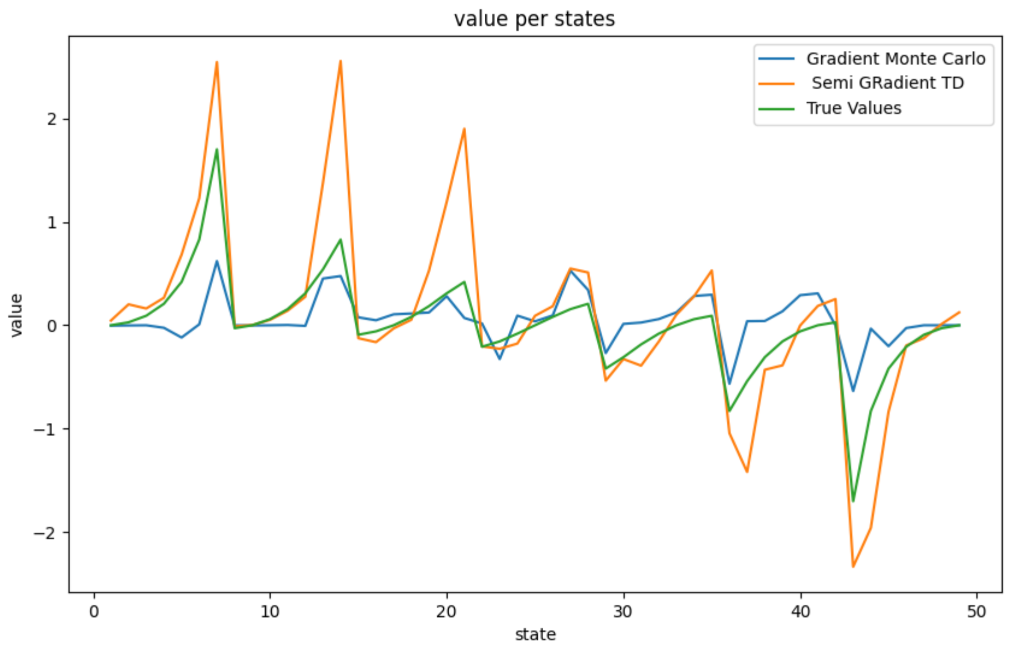

Figure 22: First Experiment: Value of each state with $\alpha = 0.5$

## References

Richard S. Sutton and Andrew G. Barto. 2018. Reinforcement Learning: An Introduction. A Bradford Book, Cambridge, MA, USA.

RL Course by David Silver - Chapter 6 howpublished = `https://www.davidsilver.uk/wp-content/uploads/2020/03/FA.pdf`,