UdS Université de Sherbrooke

Global Path Planning of the WALL-E with Reinforcement Learning Algorithms to reach EVE in an Unknown Environment

Auteur	*Email*	*Id*
Pouyan Asgharian	pouyan.asgharian@gmail.com	.....

Présenté à :
Dr. Alexandre Girard

5 avril 2022

Table des matières

1 Introduction

This project aims to test various Reinforcement Learning (RL) algorithms for the global path planning of a mobile robot. The environment is designed based on the WALL-E animation, and the tested algorithms include Q-learning, SARSA, TD(0) learning, and Double Q-learning. Temporal Difference (TD) learning is a combination of Monte Carlo ideas and dynamic programming (DP) ideas. Like Monte Carlo methods, TD methods can learn directly from raw experience without modelling the environment's dynamics. Like DP, TD methods update estimates based in part on other learned estimates without waiting for a final outcome (they bootstrap). The following sections are dedicated to explaining the environment, algorithms and output results.

2 Environment

The environment is created based on WALL-E's popular animation (Fig. 1). In this environment, the WALL-E wants to reach the goal that is the EVE robot, but there are numerous obstacles in the path that it must avoid.

FIGURE 1 – WALL-E animation.

The environment size is 15 * 15 in which each square is 40 * 40 pixels, and there are 52 obstacles inside it. Except for two robots (WALL-E and EVE), the obstacles are trees, buildings, garbage, road signs, a plant in the boot (based on animation) and a Rubik's Cube. The upside left corner (the agent starting position) is (0, 0) and going to the right and down is +X and +Y, respectively. For example, two steps to the right and one step to the down move the agent to [80, 40].

In addition, the environment is designed with *Tkinter*, a standard Python interface to the Tcl/Tk GUI toolkit. Moreover, *Pandas* library for working with tables in algorithms, *Numpy* package for scientific computing and *Matplotlib* plotting library are used. Fig. 2 shows an screenshot of the environment. The position of obstacles and the blocking area around the goal are considered in a way not to be easy for the agent to find an optimal path.

This RL environment is modelled with the Markov decision process (MDP), in which state, action and reward sets are S, A and R. The environmental dynamics would be a set of probabilities $p(s', r|s, a)$ for all states, actions and rewards. However, the testing environment is deterministic, and there are no stochastic actions.

The agent (WALL-E) can go up, down, right or left to reach the final goal in training mode. Each time the agent hits an obstacle, it will receive a -5 reward, and the system will reset to the initial point. If the robot reaches the goal, it will receive a massive reward of +100, and the reward is zero for other movements. According to animation, the interest of the WALL-E in a Rubik's cube is considered a motivation. It is not the goal, but it has a -1 reward. Solving this problem can be a good evaluation for some popular RL algorithms regarding the number of obstacles, especially around the goal, environment size and motivation between the path.

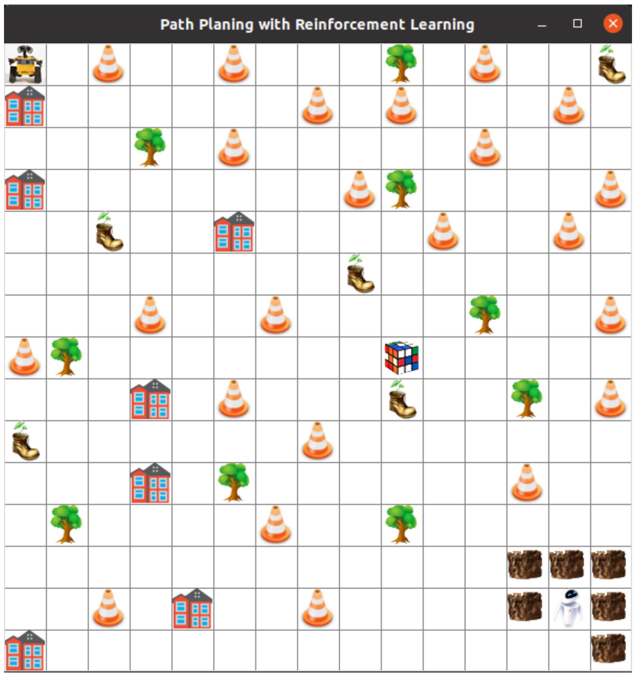


Figure 2 – Reinforcement Learning (RL) environment.

3 Q-Learning

Q-learning is an off-policy TD control algorithm that works based on the action-value function ($Q(S, A)$). In each iteration the following formula is updated :

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma max Q(S_{t+1}, a) - Q(S_t, A_t)]$$

Where α is learning rate and γ is discount factor. The Q-learning algorithm is shown below in procedural form.

The action selection mechanism in this program is $\varepsilon - greedy$. With the probability of ε, the action is chosen randomly, and with a probability of $1 - \varepsilon$, it is selected regarding maximum Q. After action selection, the agent moves in the environment to see the next state and reward. Finally, the Q-learning formula would be updated with all data collected in the

```
Q-learning (off-policy TD control) for estimating $\pi \approx \pi_*$

Algorithm parameters: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$
Initialize $Q(s, a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, arbitrarily except that $Q(terminal, \cdot) = 0$

Loop for each episode:
  Initialize $S$
  Loop for each step of episode:
    Choose $A$ from $S$ using policy derived from $Q$ (e.g., $\varepsilon$-greedy)
    Take action $A$, observe $R$, $S'$
    $Q(S, A) \leftarrow Q(S, A) + \alpha\big[R + \gamma \max_a Q(S', a) - Q(S, A)\big]$
    $S \leftarrow S'$
  until $S$ is terminal
```

environment. The mentioned loop repeats each episode until it reaches the final step. As declared before, whenever the agent reaches an obstacle or goal, the *Done* flag will active, and all the processes will restart.

For Q-learning algorithm, the value of variables are : action space = 4 = ['up', 'down', 'right', 'left'], $\alpha = 0.9$, $\gamma = 0.9$, $\varepsilon = 0.5$, decay factor = 0.999, number of episodes = 5000.

3.1 Results

After finishing training, the optimal path from initial point to Goal flag is selected as the shortest path. Also, the longest search of the agent is saved for a better comparison. The route of optimal path is shown with black ovals in Fig. 3.

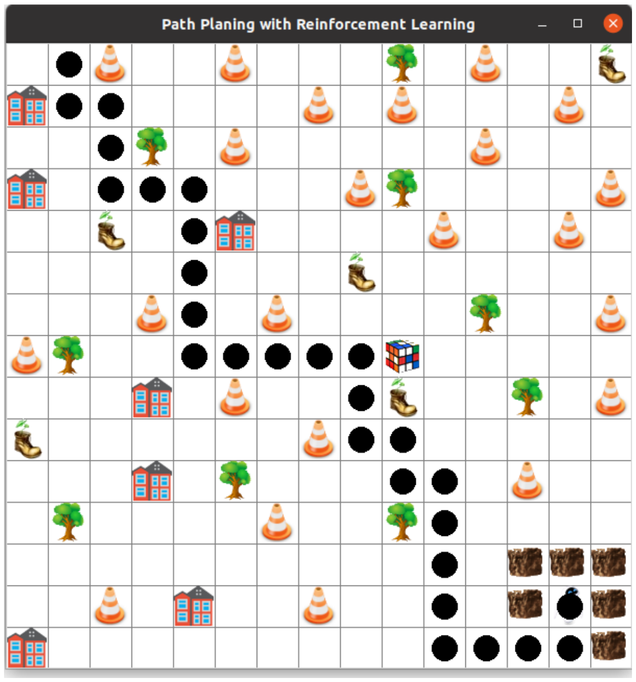


FIGURE 3 – Optimal path planning with Q-learning.

The output information is presented below. Since each square in the environment is 40 * 40 pixels, the list's arrays in the optimal path demonstrate the position of each step. In each array, the first number is X, and the second one is Y like [40.0, 0.0].

The Shortest Path : 28

The Longest Path : 1401

Optimal Path :
[[40.0, 0.0], [40.0, 40.0], [80.0, 40.0], [80.0, 80.0], [80.0, 120.0], [120.0, 120.0], [160.0, 120.0], [160.0, 160.0], [160.0, 200.0], [160.0, 240.0], [160.0, 280.0], [200.0, 280.0], [240.0, 280.0], [280.0, 280.0], [320.0, 280.0], [320.0, 320.0], [320.0, 360.0], [360.0, 360.0], [360.0, 400.0], [400.0, 400.0], [400.0, 440.0], [400.0, 480.0], [400.0, 520.0], [400.0, 560.0], [440.0, 560.0], [480.0, 560.0], [520.0, 560.0], [520.0, 520.0]]

Final Q-table :

States	0	1	2	3
[40.0, 0.0]	0.000000	6.461082	-5.000000	0.000000
[40.0, 40.0]	0.000000	0.000000	7.178980	-5.000000
[80.0, 40.0]	-5.000000	7.976644	0.000000	0.000000
[80.0, 80.0]	0.000000	8.862938	-5.000000	0.000000
[80.0, 120.0]	0.000000	-5.000000	9.847709	0.000000
[120.0, 120.0]	-5.000000	0.000000	10.941899	7.976644
[160.0, 120.0]	0.000000	12.157665	0.000000	0.000000
[160.0, 160.0]	9.847709	13.508517	-5.000000	0.000000
[160.0, 200.0]	0.000000	15.009464	0.000000	0.000000
[160.0, 240.0]	0.000000	16.677182	0.000000	-4.950000
[160.0, 280.0]	0.000000	0.000000	18.530202	0.000000
[200.0, 280.0]	0.000000	-4.950000	20.589113	0.000000
[240.0, 280.0]	-4.999500	0.000000	22.876792	0.000000
[280.0, 280.0]	0.000000	0.000000	25.418658	0.000000
[320.0, 280.0]	0.000000	28.242954	-0.900000	0.000000
[320.0, 320.0]	0.000000	31.381060	-4.500000	0.000000
[320.0, 360.0]	0.000000	0.000000	34.867844	-4.500000
[360.0, 360.0]	-4.500000	38.742049	0.000000	0.000000
[360.0, 400.0]	0.000000	-4.500000	43.046721	0.000000
[400.0, 400.0]	0.000000	47.829690	0.000000	0.000000
[400.0, 440.0]	0.000000	53.144100	0.000000	-4.500000
[400.0, 480.0]	0.000000	59.049000	0.000000	43.046717
[400.0, 520.0]	0.000000	65.610000	0.000000	0.000000
[400.0, 560.0]	0.000000	0.000000	72.900000	0.000000
[440.0, 560.0]	0.000000	0.000000	81.000000	0.000000
[480.0, 560.0]	-4.950000	0.000000	90.000000	0.000000
[520.0, 560.0]	100.000000	0.000000	-4.500000	0.000000

Full Q-table : [176 rows x 4 columns]

States	0	1	2	3
[0.0, 0.0]	0.000000	-5.000000	5.814974	0.000000
Obstacle	0.000000	0.000000	0.000000	0.000000
[40.0, 0.0]	0.000000	6.461082	-5.000000	0.000000
....	....	....	....	....
[520.0, 0.0]	0.000000	-4.500000	-4.500000	0.000000

Besides, the following figures (Fig 4, 5, 6) represent the total value, total steps and total reward for 5000 episodes. After 1000 episodes, the Q-learning algorithms found the optimal path. The reward plot shows that before reaching +100 (goal position), it is mainly -5 and -1 (due to touching Rubic's cube location) in some episodes. The number of steps before convergence is higher because of the agent's searching, but after that, it decreases. The value plot behaves similarly to reward, and it is zero before reaching the goal due to restarts. It can be found that after reaching the goal, the agent will stop searching and consider the path to the goal as an optimal and shortest one.

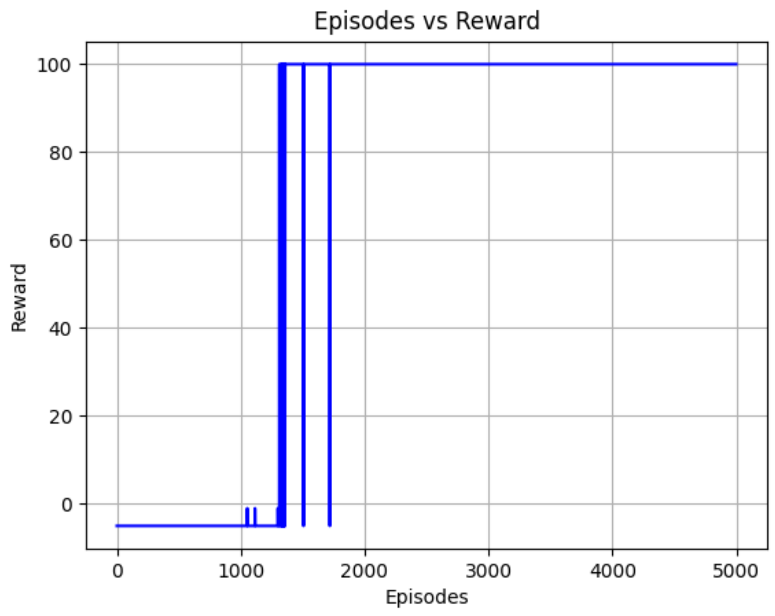


FIGURE 4 – Total reward in Q-learning.

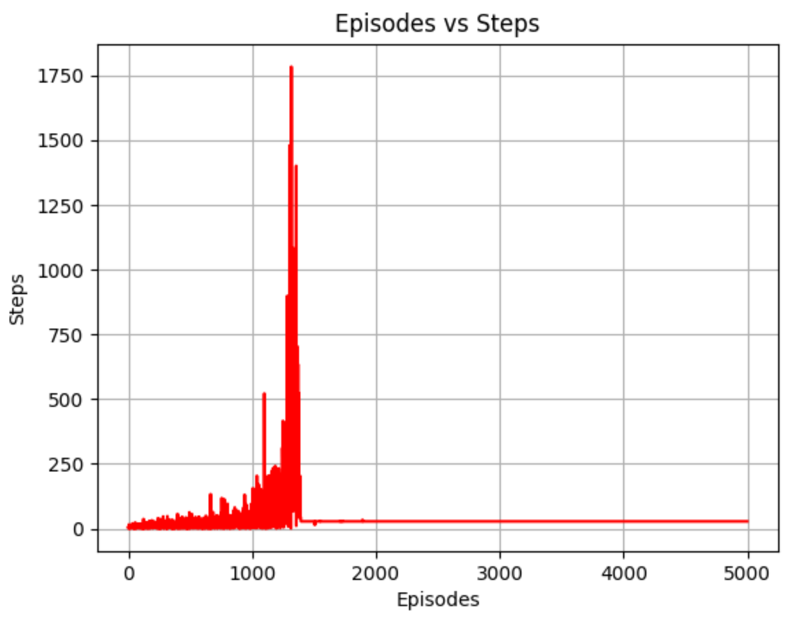


FIGURE 5 – Total steps in Q-learning.

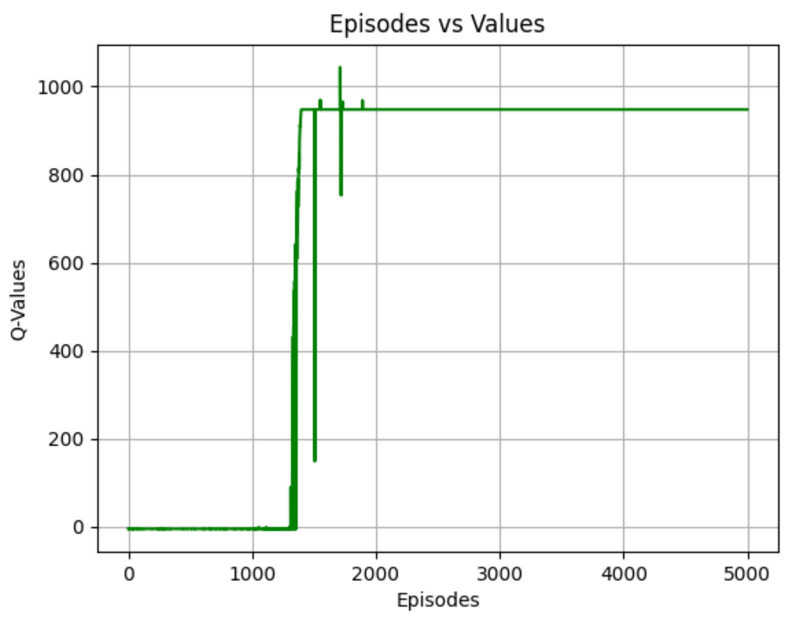


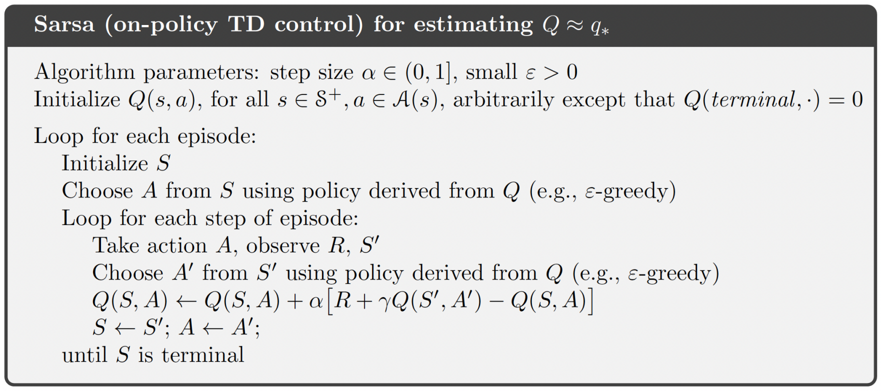
FIGURE 6 – Total values in Q-learning.

4 SARSA

SARSA is an on-policy TD control method that has common features with Q-learning. In each iteration, the following formula will be updated :

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)]$$

The convergence properties of the Sarsa algorithm depend on the nature of the policy's dependence on Q. The general form of the Sarsa control algorithm is given in the box below. As it is understandable, the SARSA needs the next action ($A_{t+1} = A^{'}$) to update the Q table, and hence it chooses the next action from the next state ($S_{t+1} = S^{'}$).



```
Sarsa (on-policy TD control) for estimating $Q \approx q_*$

Algorithm parameters: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$
Initialize $Q(s, a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, arbitrarily except that $Q(terminal, \cdot) = 0$

Loop for each episode:
    Initialize $S$
    Choose $A$ from $S$ using policy derived from $Q$ (e.g., $\varepsilon$-greedy)
    Loop for each step of episode:
        Take action $A$, observe $R$, $S'$
        Choose $A'$ from $S'$ using policy derived from $Q$ (e.g., $\varepsilon$-greedy)
        $Q(S, A) \leftarrow Q(S, A) + \alpha\big[R + \gamma Q(S', A') - Q(S, A)\big]$
        $S \leftarrow S'$; $A \leftarrow A'$;
    until $S$ is terminal
```

The action selection mechanism is epsilon greedy, and all procedures are the same as Q-learning except for episodes in which SARSA requires more time to find the optimal path. For SARSA algorithm, the value of variables are : action space = 4 = ['up', 'down', 'right', 'left'], $\alpha = 0.9$, $\gamma = 0.9$, $\varepsilon = 0.5$, decay factor = 0.999, number of episodes = 50000.

4.1 Results

From the output results it can be found that after approximately 25000 episodes, the agent could find a path to the goal location. Fig. 7 shows the optimal route with black ovals.

The number of steps in SARSA and Q-learning is the same (28 steps), but they selected different paths. Moreover, contrary to Q-learning, SARSA continues learning and searching until the end of the episode. The output data and Q-tables are presented below.

The Shortest Path : 28

The Longest Path : 550

Optimal Path :
[[40.0, 0.0], [40.0, 40.0], [40.0, 80.0], [40.0, 120.0], [40.0, 160.0], [40.0, 200.0], [40.0, 240.0], [80.0, 240.0], [80.0, 280.0], [120.0, 280.0], [160.0, 280.0], [160.0, 320.0], [160.0, 360.0], [200.0, 360.0], [240.0, 360.0], [240.0, 400.0], [280.0, 400.0], [320.0, 400.0], [320.0, 440.0], [320.0, 480.0],

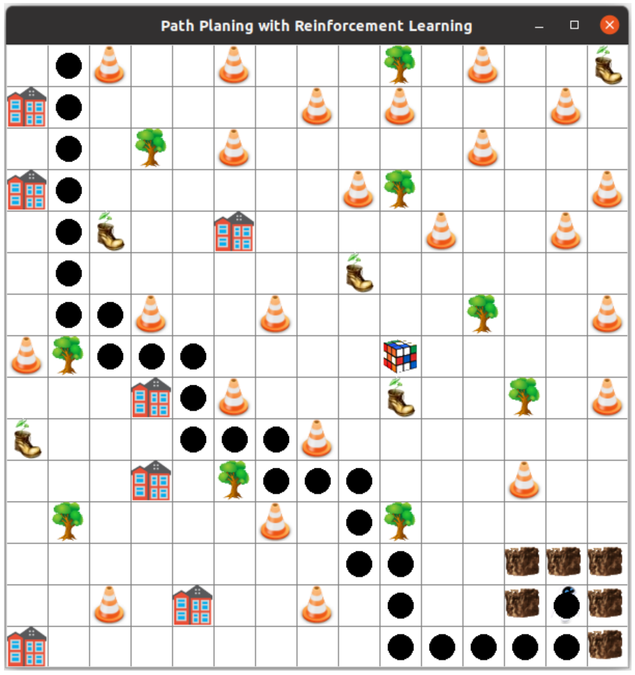


FIGURE 7 – Optimal path planning with SARSA.

[360.0, 480.0], [360.0, 520.0], [360.0, 560.0], [400.0, 560.0], [440.0, 560.0], [480.0, 560.0], [520.0, 560.0], [520.0, 520.0]]

Final Q-table :

States	0	1	2	3
[40.0, 0.0]	-4.104407	2.099550	-5.000000	-2.055044
[40.0, 40.0]	-3.302516	1.967389	-4.061254	-5.000000
[40.0, 80.0]	-4.119704	1.462201	-0.919095	-0.012449
[40.0, 120.0]	-3.295571	0.998683	1.750480	-5.000000
[40.0, 160.0]	-2.239626	0.895676	-5.000000	-3.647892
[40.0, 200.0]	-0.719129	-3.280719	0.391534	-1.899991
[40.0, 240.0]	-0.233413	-5.000000	9.847704	-3.284249
[80.0, 240.0]	-4.000866	1.387104	-5.000000	-4.050001
[80.0, 280.0]	-2.664114	-3.163169	0.324364	-5.000000
[120.0, 280.0]	-5.000000	-5.000000	0.170468	-1.978908
[160.0, 280.0]	-3.439421	0.036780	-3.314143	-0.500884
[160.0, 320.0]	-2.579885	-2.332648	-5.000000	-5.000000
[160.0, 360.0]	-0.973352	-0.428563	-3.493924	-4.050894
[200.0, 360.0]	-5.000000	-5.000000	-2.542879	15.517152
[240.0, 360.0]	-3.282946	-1.290420	-5.000000	-4.048767
[240.0, 400.0]	-4.020465	-4.999995	1.036072	-5.000000
[280.0, 400.0]	-5.000000	-1.160020	7.942310	-4.110057
[320.0, 400.0]	-2.777621	16.501423	-4.033209	-2.777935
[320.0, 440.0]	-3.287799	26.688280	-5.000000	-3.087450
[320.0, 480.0]	-1.166957	1.711174	7.072903	-0.391565
[360.0, 480.0]	-5.000000	43.939945	0.138748	0.238742
[360.0, 520.0]	4.678612	5.156711	58.510487	4.770994
[360.0, 560.0]	5.121811	4.392585	3.671136	6.237530
[400.0, 560.0]	59.049000	53.883428	72.220537	7.833147
[440.0, 560.0]	50.404889	13.880242	81.000000	3.740890
[480.0, 560.0]	-5.000000	-2.835820	90.000000	4.278974
[520.0, 560.0]	100.000000	90.000000	-5.000000	49.840349

Full Q-table : [176 rows x 4 columns]

States	0	1	2	3
[0.0, 0.0]	-4.083781	-5.000000	2.638752	-4.418550
Obstacle	0.000000	0.000000	0.000000	0.000000
[40.0, 0.0]	-4.104407	2.099550	-5.000000	-2.055044
...	...	...	...	...
[520.0, 400.0]	-2.954453	-2.130813	-0.287858	-4.999500

Total rewards, total steps and total values are represented in Fig. 8, 9 and 10, respectively. It can be seen that before reaching the goal, the reward is mainly -5 and, in some episodes is -1. The interesting note about SARSA is its searching even after reaching the goal. SARSA's value level is higher than Q-learning, and continuous searching is one reason. Moreover, in Final Q-table, the values demonstrate the broad searching and less directed SARSA algorithm than Q-learning. From plots, it can be found that While SARSA's learning is slower than Q-learning, it is more confident about previous mistakes and found paths.

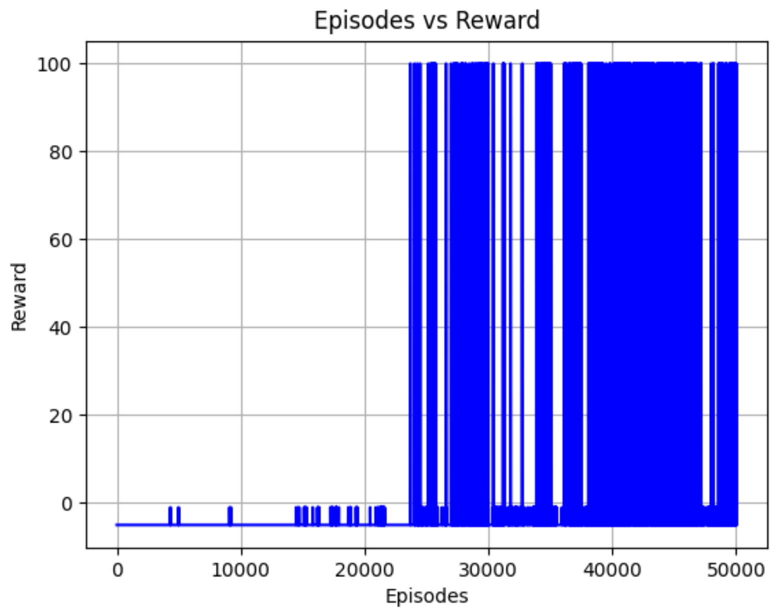


FIGURE 8 – Total reward in SARSA.

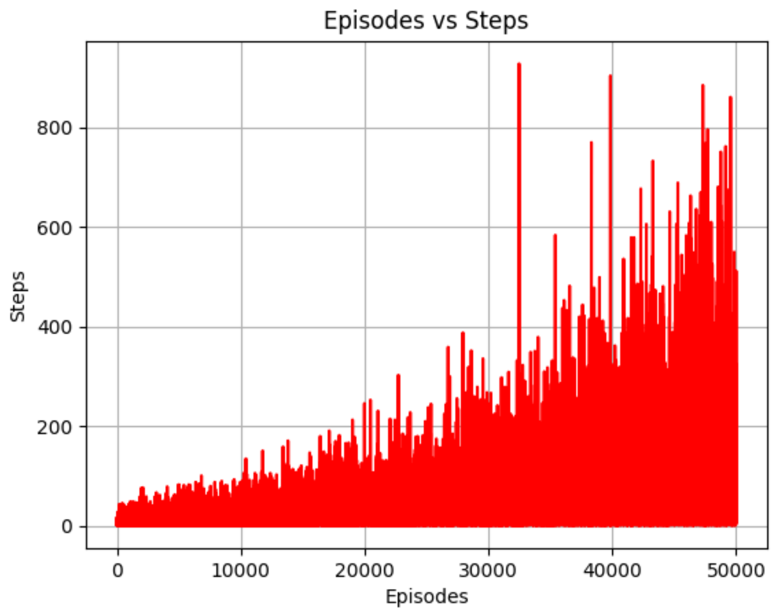


FIGURE 9 – Total steps in SARSA.

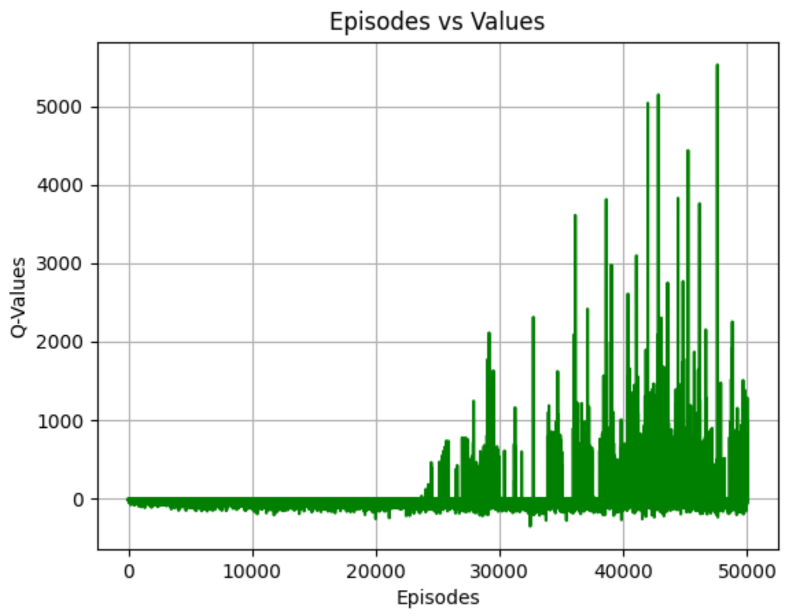


FIGURE 10 – Total values in SARSA.

5 TD(0)

TD method uses experience to solve the prediction problem. Given some experiences following a policy π, this method updates its estimate V for the non-terminal states S_t occurring in that experience. Fig. 11 shows the working procedure of TD(0) or one-step TD with its equation. Also, The box after that specifies TD(0) completely in procedural form. In TD(0) state-value function ($V(s)$) is used instead of action-value function ($Q(s, a)$).

In this algorithm, the actions are selected regarding the policy. In each state, the agent randomly chooses an action regarding a simple policy. When going right hits an obstacle, the agent will choose other actions randomly, and this situation is the same for other directions. For TD(0) algorithm, the value of variables are as follows : action space = 4 = ['up', 'down', 'right', 'left'], $\alpha = 0.9$, $\gamma = 0.9$, $\varepsilon = 0.5$, decay factor = 0.999, number of episodes = 50000.

5.1 Results

In this section, outputs of one-step TD are presented. Since the actions are selected based on the policy, the agent reaches the goal in all the steps, and the number of steps per episodes are somehow the same. Due to the undirected search, the state value is not high, and in many steps, it is negative. The output data for the final path is presented below and the optimal path is represented by black ovals in fig. 12.

The Shortest Path : 95

The Longest Path : 336

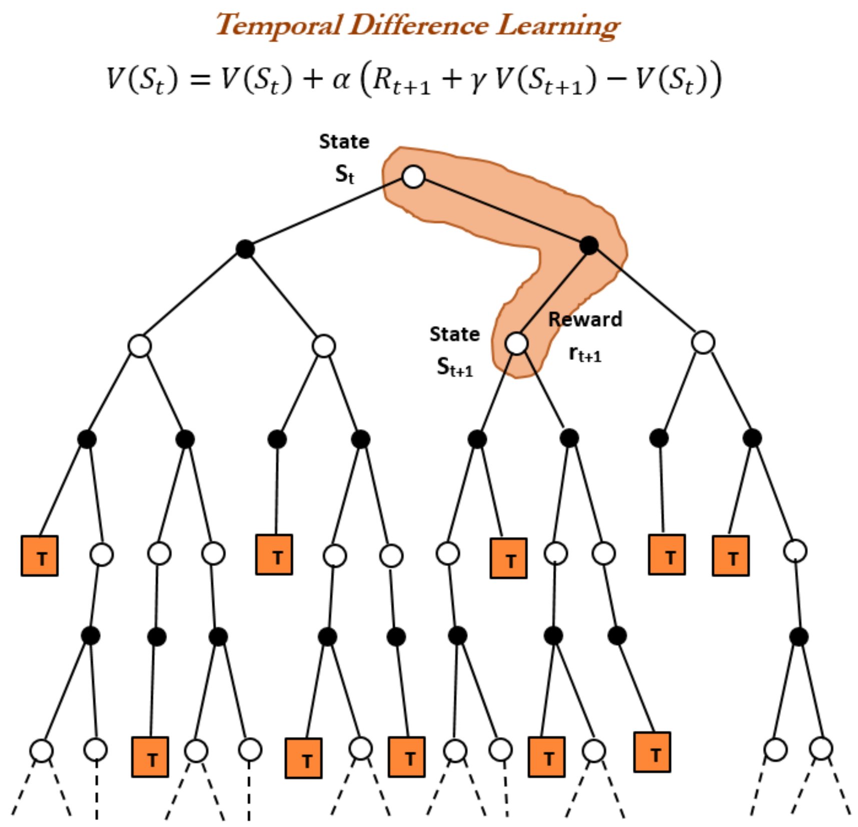

Temporal Difference Learning
$V(S_t) = V(S_t) + \alpha \left(R_{t+1} + \gamma V(S_{t+1}) - V(S_t)\right)$
State
S_t
State
S_{t+1}
Reward
r_{t+1}
T

FIGURE 11 – TD method.

Tabular TD(0) for estimating v_π

```
Input: the policy $\pi$ to be evaluated
Algorithm parameter: step size $\alpha \in (0, 1]$
Initialize $V(s)$, for all $s \in \mathcal{S}^+$, arbitrarily except that $V(terminal) = 0$

Loop for each episode:
    Initialize $S$
    Loop for each step of episode:
        $A \leftarrow$ action given by $\pi$ for $S$
        Take action $A$, observe $R$, $S'$
        $V(S) \leftarrow V(S) + \alpha\big[R + \gamma V(S') - V(S)\big]$
        $S \leftarrow S'$
    until $S$ is terminal
```

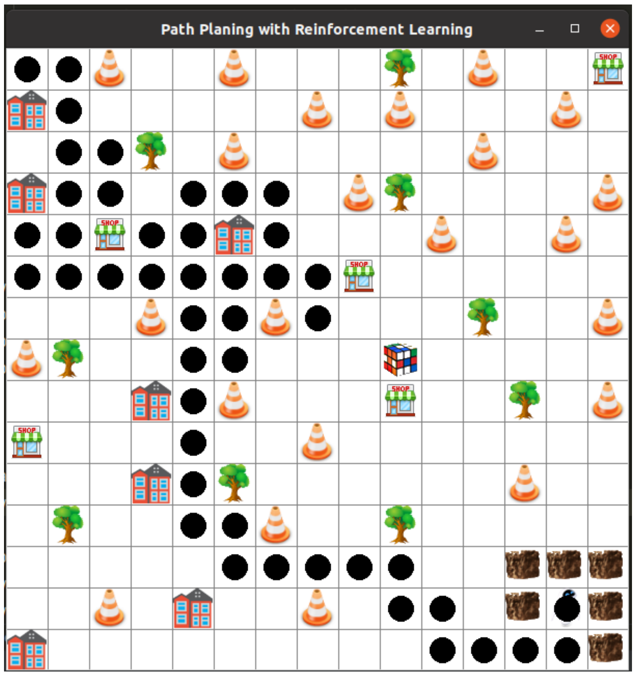


FIGURE 12 – Optimal path planning with TD(0).

Optimal Path :

[[0.0, 0.0], [40.0, 0.0], [0.0, 0.0], [0.0, 0.0], [0.0, 0.0], [0.0, 0.0], [0.0, 0.0], [0.0, 0.0], [0.0, 0.0], [0.0, 0.0], [0.0, 0.0], [40.0, 0.0], [40.0, 0.0], [0.0, 0.0], [40.0, 0.0], [0.0, 0.0], [40.0, 0.0], [40.0, 0.0], [40.0, 40.0], [40.0, 80.0], [40.0, 120.0], [40.0, 80.0], [40.0, 120.0], [40.0, 80.0], [40.0, 120.0], [40.0, 80.0], [80.0, 80.0], [80.0, 120.0], [40.0, 120.0], [40.0, 160.0], [0.0, 160.0], [0.0, 200.0], [0.0, 160.0], [0.0, 160.0], [0.0, 200.0], [40.0, 200.0], [80.0, 200.0], [120.0, 200.0], [120.0, 160.0], [160.0, 160.0], [160.0, 120.0], [200.0, 120.0], [240.0, 120.0], [240.0, 160.0], [240.0, 200.0], [200.0, 200.0], [200.0, 240.0], [200.0, 200.0], [200.0, 240.0], [160.0, 240.0], [160.0, 280.0], [160.0, 240.0], [160.0, 280.0], [160.0, 240.0], [160.0, 280.0], [160.0, 240.0], [160.0, 200.0], [120.0, 200.0], [80.0, 200.0], [120.0, 200.0], [160.0, 200.0], [200.0, 200.0], [240.0, 200.0], [280.0, 200.0], [280.0, 240.0], [280.0, 200.0], [240.0, 200.0], [200.0, 200.0], [200.0, 240.0], [200.0, 280.0], [160.0, 280.0], [160.0, 320.0], [160.0, 360.0], [160.0, 400.0], [160.0, 440.0], [200.0, 440.0], [200.0, 480.0], [240.0, 480.0], [280.0, 480.0], [320.0, 480.0], [360.0, 480.0], [360.0, 520.0], [400.0, 520.0], [400.0, 560.0], [440.0, 560.0], [480.0, 560.0], [440.0, 560.0], [480.0, 560.0], [480.0, 560.0], [480.0, 560.0], [480.0, 560.0], [520.0, 560.0], [520.0, 560.0], [520.0, 560.0], [520.0, 520.0]]

Final V-table :

States	Values
[0.0, 0.0]	-0.048587
[40.0, 0.0]	-0.232032
[40.0, 40.0]	-1.110985
[40.0, 80.0]	-0.677959
[40.0, 120.0]	-0.082712
[80.0, 80.0]	-0.580085
[80.0, 120.0]	-0.226284
[40.0, 160.0]	-0.109950
[0.0, 160.0]	-0.126974
[0.0, 200.0]	-0.119014
[40.0, 200.0]	-0.097870
[80.0, 200.0]	-0.199502
[120.0, 200.0]	-0.271317
[120.0, 160.0]	-0.133922
[160.0, 160.0]	-0.299506
[160.0, 120.0]	-3.808410
[200.0, 120.0]	-1.061209
[240.0, 120.0]	-3.797218
[240.0, 160.0]	-0.847080
[240.0, 200.0]	-0.352839
[200.0, 200.0]	-0.517903
[200.0, 240.0]	-0.222678
[160.0, 240.0]	-2.248170
[160.0, 280.0]	-0.168093
[160.0, 200.0]	-0.177441
[280.0, 200.0]	-0.182979
[280.0, 240.0]	-0.450525
[200.0, 280.0]	-0.207689
[160.0, 320.0]	-4.799294
[160.0, 360.0]	-2.910989
[160.0, 400.0]	-4.860101
[160.0, 440.0]	-1.272738
[200.0, 440.0]	-4.681947
[200.0, 480.0]	-3.562581
[240.0, 480.0]	-3.315708
[280.0, 480.0]	-0.111897
[320.0, 480.0]	0.119484
[360.0, 480.0]	0.306290
[360.0, 520.0]	0.335131
[400.0, 520.0]	0.408170
[400.0, 560.0]	3.452500
[440.0, 560.0]	6.391877
[480.0, 560.0]	25.508254
[520.0, 560.0]	65.003347

Full Q-table : [176 rows x 1 columns]

States	Values
[0.0, 0.0]	-0.048587
[40.0, 0.0]	-0.232032
[40.0, 40.0]	-1.110985
...	...
Goal	0.000000
[520.0, 0.0]	-3.951864
[560.0, 160.0]	-4.993713

Total rewards, total steps and total values are represented in Fig. 13, 14 and 15, respectively. It can be seen that the agent hits the goal in whole the episode due to using a predefined policy. In addition, the number of steps and values is approximately the same in whole the run-time. The V-values are mainly negative, and in some steps, they are positive.

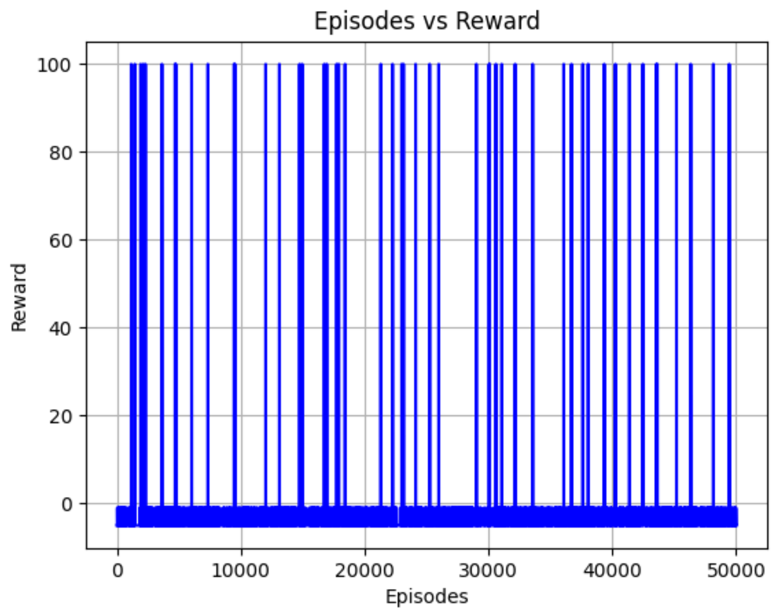


Figure 13 – Total reward in TD(0).

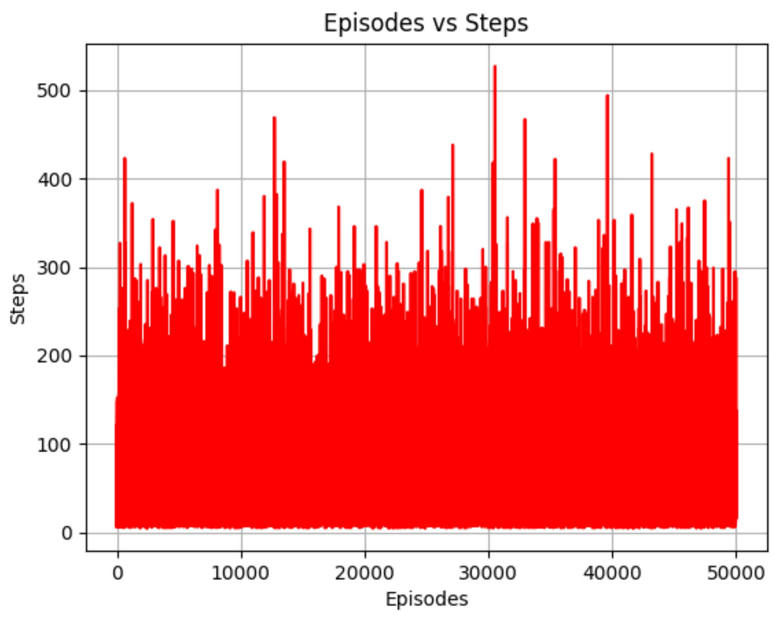


FIGURE 14 – Total steps in TD(0).

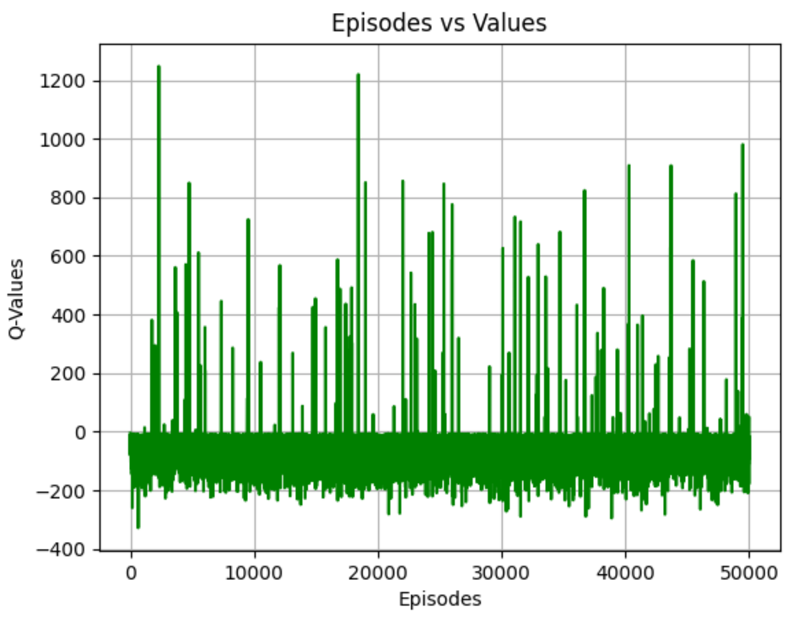


FIGURE 15 – Total values in TD(0).

6 Double Q-learning

The idea of double learning extends naturally to algorithms for full MDPs. For example, the double learning algorithm analogous to Q-learning, called Double Q-learning, divides the time steps in two, perhaps by flipping a coin on each step. If the coin comes up heads, the update is :

$$Q_1(S_t, A_t) \leftarrow Q_1(S_t, A_t) + \alpha[R_{t+1} + \gamma Q_2(S_{t+1}, argmax Q_1(S_{t+1}, a)) - Q_1(S_t, A_t)]$$

Although we learn two estimates, only one estimate is updated on each play ; double learning doubles the memory requirements but does not increase the amount of computation per step. If the coin comes up tails, then the same update is done with Q1, and Q2 is switched so that Q2 is updated. The two approximate value functions are treated entirely symmetrically. The behaviour policy can use both action-value estimates. A complete algorithm for Double Q-learning is given in the box below.

Double Q-learning, for estimating $Q_1 \approx Q_2 \approx q_*$

```
Algorithm parameters: step size $\alpha \in (0,1]$, small $\varepsilon > 0$
Initialize $Q_1(s,a)$ and $Q_2(s,a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, such that $Q(terminal, \cdot) = 0$

Loop for each episode:
  Initialize $S$
  Loop for each step of episode:
    Choose $A$ from $S$ using the policy $\varepsilon$-greedy in $Q_1 + Q_2$
    Take action $A$, observe $R$, $S'$
    With 0.5 probabilility:
      $Q_1(S,A) \leftarrow Q_1(S,A) + \alpha\Big(R + \gamma Q_2\big(S', \text{argmax}_a Q_1(S',a)\big) - Q_1(S,A)\Big)$
    else:
      $Q_2(S,A) \leftarrow Q_2(S,A) + \alpha\Big(R + \gamma Q_1\big(S', \text{argmax}_a Q_2(S',a)\big) - Q_2(S,A)\Big)$
    $S \leftarrow S'$
  until $S$ is terminal
```

The action selection mechanism is $\varepsilon - greedy$ in which the action is selected from $Q_1 + Q_2$. All procedures are the same as Q-learning for going to the next state and receiving a reward, except updating two Q tables with 50% probability. For Double Q-learning algorithm, the value of variables are : action space = 4 = ['up', 'down', 'right', 'left'], $\alpha = 0.9$, $\gamma = 0.9$, $\varepsilon = 0.5$, decay factor = 0.999, probability = 0.5, number of episodes = 50000.

6.1 Results

to be completed ....

7 Cliff Walking problem

This problem is derived from "Reinforcement Learning : An Introduction" book by Andrew Barto and Richard S. Sutton. The environment is similar to Fig. 16. This is a standard undiscounted, episodic task, with start and goal states and the usual actions causing movement up, down, Optimal path right, and left. The reward is -1 on all transitions except those into the region marked The Cliff. Stepping into this region incurs a reward of -100 and sends the agent instantly back to the start.

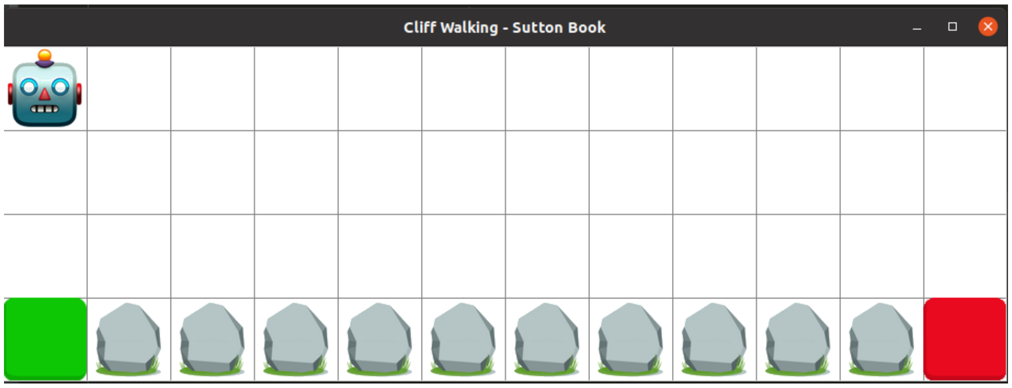


FIGURE 16 – Cliff walking environment.

The starting point is a green square, and the goal is reaching to the red one. Stones show the cliffs, and the agent is a robot that wants to plan an optimal path. In order to have a comprehensive analyse, all the mentioned algorithms (Q-learning, SARSA and Double Q-learning) are tested in this environment. The environment and algorithms variables are as follow : action space = 4 = ['up', 'down', 'right', 'left'], $\alpha = 0.9$, $\gamma = 0.9$, $\varepsilon = 0.1$[1], decay factor = 0.999, number of episodes = 500.

The output routes of all algorithms are shown in Fig. 17. Q-learning, SARSA and Double Q-learning paths are represented by blue, pink and orange ovals for better visualisation, respectively.

According to Fig. 17 it can be seen that Q-learning chooses the shortest path beside cliffs. On the other hand, SARSA takes the action selection into account and learns the longer but safer path through the upper part of the grid. The randomised performance in Double Q-learning is higher than in others.

Total rewards, total steps and total values for three algorithms are shown in Fig. 18, 19 and 20, respectively. It can be seen that the reward value of Q-learning and SARSA is better than Double Q-learning, but it is because of more randomness of it ! The values that SARSA obtain are more than Q-learning due to safer path planning.

1. Since Double Q-learning always selects the safest path, it is better to increase the epsilon value in order to reduce the run time. Otherwise, it will search a lot, especially on the upper side, far from cliffs, to find the optimal path.

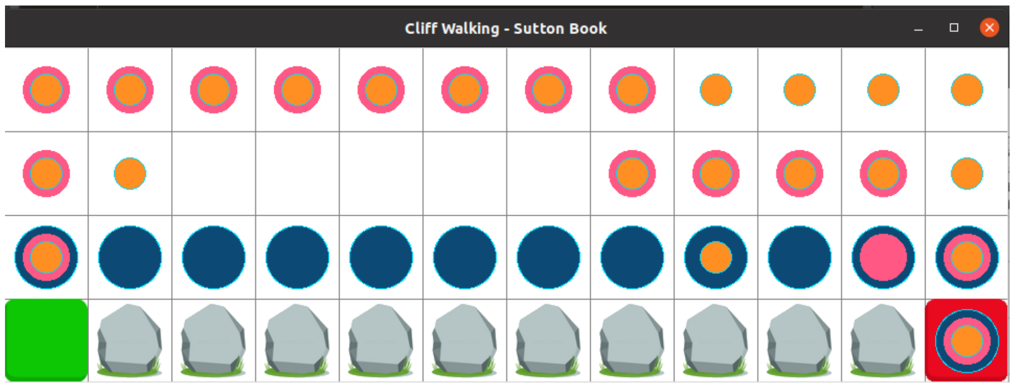


FIGURE 17 – Optimal paths in cliff walking problem.

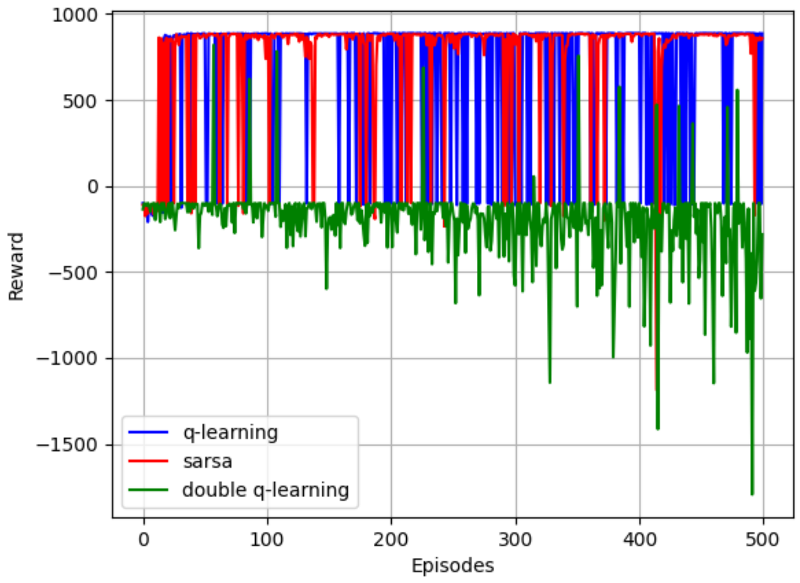


FIGURE 18 – Total reward in cliff walking.

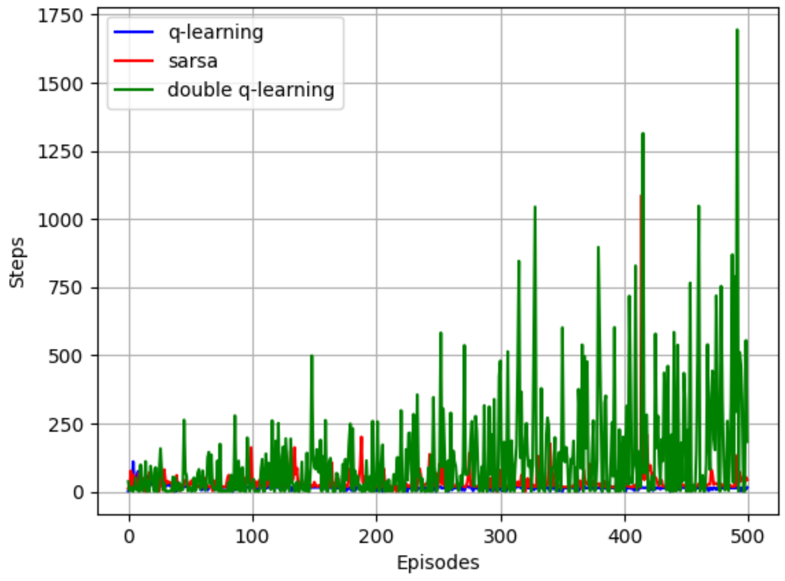


FIGURE 19 – Total steps in cliff walking.

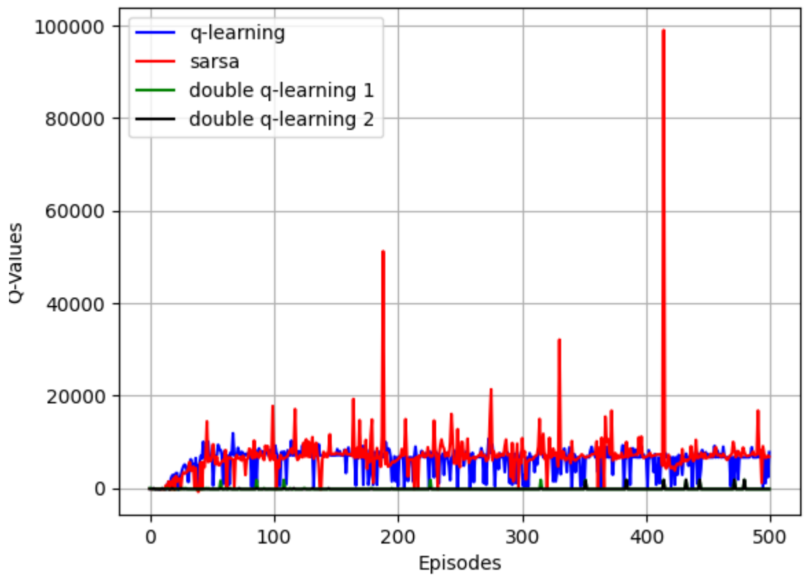


FIGURE 20 – Total values in cliff walking.