Solutions for HW6

Yunhai Han
Department of Mechanical and Aerospace Engineering
University of California, San Diego
y8han@eng.ucsd.edu

May 29, 2020

Contents

1 Problem 1

1.1 problem formulation

The cliffGrid.py environment consists of a 12×4 grid world, and has a "cliff" consisting of the lose (or cliff) states $(2,1), \ldots, (11,1)$. The START state and the EXIT state are respectively (1,1) and (12,1) Transitions in this environment are given by nextState, which implements deterministically the usual up, down, left, right to a legal state in the grid, and, in addition, when the next state belongs to the cliff, nextState drives the agent back to the start position. A terminal control "0" is allowable at the goal state so the agent can remain in it. The cost function is given in cliffGridcost. By means of this function, all up, down, left, right legal transitions get a cost of +1 , except for transitions into the cliff. The cost of the terminal action 0 is zero. You have to fill out the template learning.py to implement SARSA and the Q-learning functions.

1.2 part 1

1.2.1 formulation

Implement the SARSA algorithm using a fixed $\epsilon_k \equiv \epsilon = 0.5$ value. Follow the template of SARSA in the slides (page 25/32) but make use of $M = 5$, and episodes of lengths $T_0 = 10, T_1 = 40, T_2 = 100$ $T_3 = 100$ and $T_4 = 250$, respectively in each iteration of the M loop. In the initial trajectory, take the start state to be START, then at the next iterations of the M loop, generate a new episode from the the state you reached in previous iteration (unless it is the EXIT state, in which case repeat from START state or any other state). Plot the estimated cumulative cost of the START state to the EXIT state by using the Q function as time progresses from $t = 0$ to $T = 500$ (use all updates of the Q function obtained in all iterations). Note that the start state should be visited frequently as we fall into cliff states, so the Q function at states $Q(\text{ start}, u)$ will be updated frequently. As for initialization, use $Q_0(i,u) = +100$ for all i except for the goal state, where you can use $Q_0(\text{ goal}, 0) = 0$. Are optimal paths close or far from the cliff after algorithm termination or not? How about the obstacle?

1.2.2 solution

In the slides(page 25/32), the SARSA algorithm could be written as:

Input: A (partially known) Markov Decision Process, an exploration sequence $\{\varepsilon_k\}$ Output: An approximation of $Q_M(i,u)$ for each i and u

```
1: Initialize $Q_0(i,u), N(i) = 0, \alpha_0(i) = 1$, for each $i$, and all $u$
2: for $k$ in $0, \ldots, M-1$
3 :    Obtain an $\varepsilon_{k-\text{ greedy policy from }} Q_k$, call it $\pi_k$
4:     Generate a $T$ -long episode $\rho_0^{(k)}$ from $\pi_k$
5 :    for each $(x_t, u_t, x_{t+1}, u_{t+1})$ in $\rho_0^{(k)}$
```

$$Q_{k+1}(x_t,u_t) = Q_k(x_t,u_t) +$$
$$\alpha_k(x_t)\left[\ell(x_t,x_{t+1}) + \gamma Q_k(x_{t+1},u_{t+1}) - Q_k(x_t,u_t)\right]$$
$$N(x_{t+1}) = N(x_{t+1}) + 1, \quad \alpha_{k+1}(x_{t+1}) = \left(N(x_{t+1}) + 1\right)^{-1}$$

From piazaa, professor suggested to use a small γ, so in the codes, I set it as 0.1. From the title, we know that $M = 5$ and during each iteration, the length of episodes are differen-t(10,40,100,100,250).

The results are shown below:

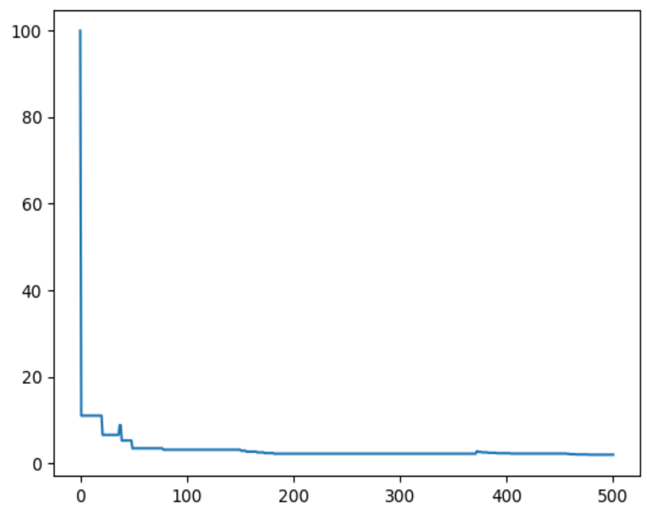

Figure 1: The estimated cumulative cost of the **START** state

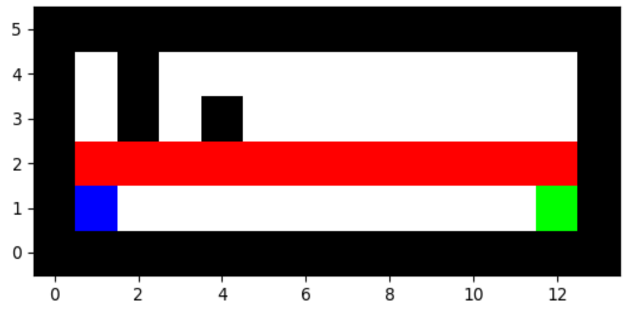

Figure 2: The optimal path towards **EXIT**

There is one point that it is not guaranteed for us to find the feasible path. I run the codes for several times and in most cases it can not find such path. The reason is simple: due to the existence of the motion noise, if the robot falls into cliff states, it is reset at the start state and has to give an another try. In this example, $\epsilon_k \equiv \epsilon = 0.5$ and it means it is very possible for the robot to take the wrong actions. Besides, since the initial Q values are generated randomly, we could not expect the initial control policy are good enough for the quick convergence, which would also lead to bad performance in limited times.

Every time if we could find such a path, the path is always the same: the robot moves along the cliff states and never falls into them. There are small differences for the estimated cumulative cost and I show two other plots:

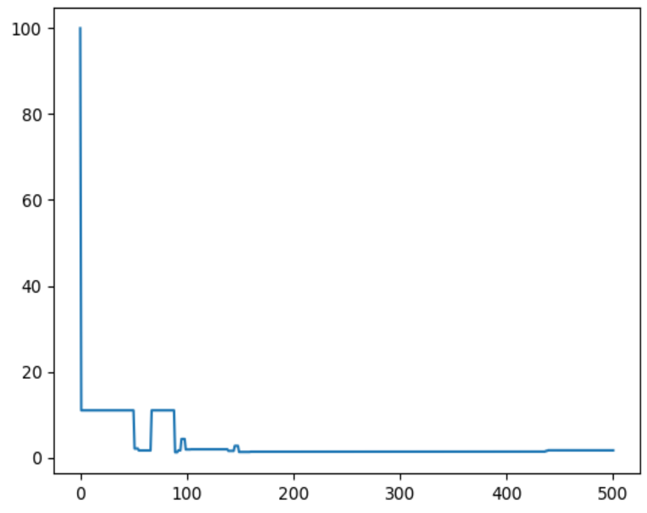

Figure 3: The estimated cumulative cost of the **START** state

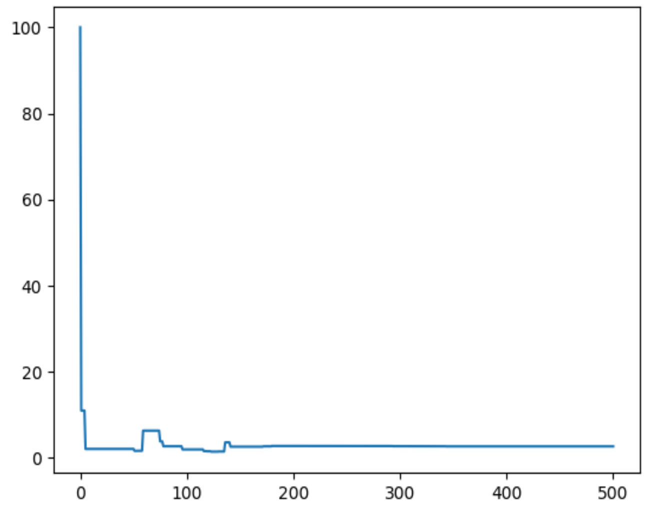

Figure 4: The estimated cumulative cost of the **START** state

To sum up, the optimal paths are close to the cliff and the robot would not run into obstacles.

1.3 part 2

1.3.1 formulation

Now implement the SARSA algorithm using a sequence of 4ϵ values: $\epsilon_0 = 0.5, \epsilon_1 = 0.3, \epsilon_2 = 0.2$ and $\epsilon_3 = 0.1$. However, instead of following the template in the slides exactly make use of $M = 1000$ and $T = 2$ long episodes. This version of the algorithm will allow you to use more directly an updated Q function, which is a widely used version of SARSA (see [1]). Split $M = 1000$ into sequences of 250 after which you can update to smaller ϵ_k. Plot the value of the estimated cumulative cost from the START as time t evolves from $t = 0$ to $T = 1000$. Do you observe any differences with the previous solution? Is the optimal path any different? What happens if you keep lowering ϵ with longer M loops?

1.3.2 solution

In this part, I only to make a little modification on the codes I used before. To be specific, I fixed the length of episodes as 2 and split $M = 1000$ into sequences of 250 after which I update to smaller ϵ_k(0.5,0.3,0.2,0.1).

The results are different from the previous ones:

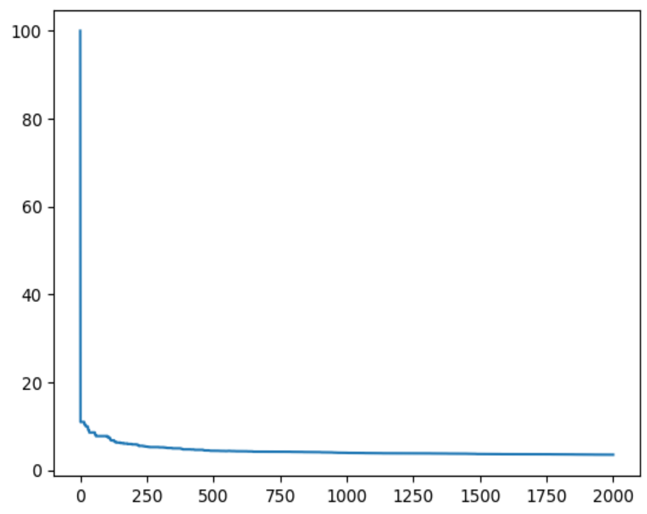

Figure 5: The estimated cumulative cost of the **START** state

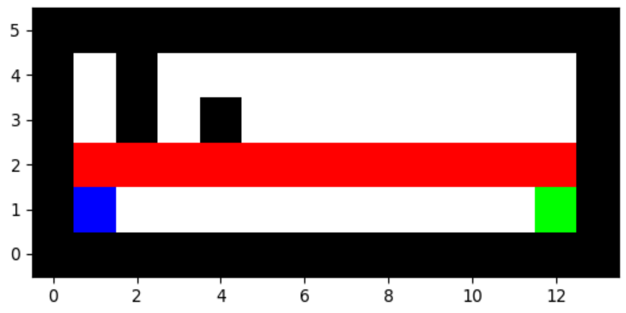

Figure 6: The optimal path towards **EXIT**

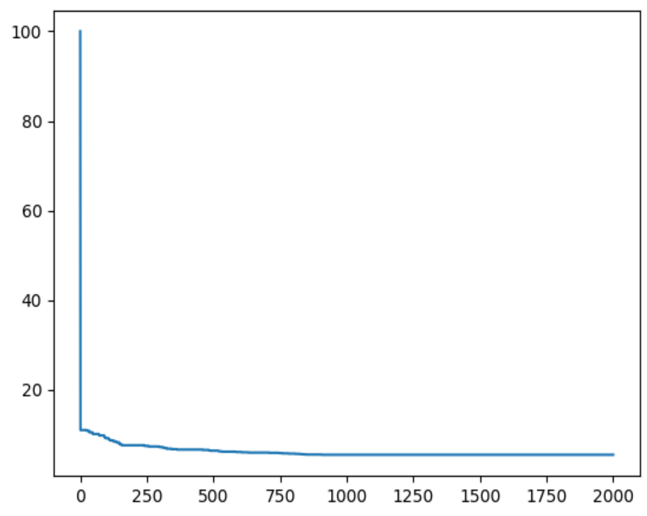

Figure 7: The estimated cumulative cost of the **START** state

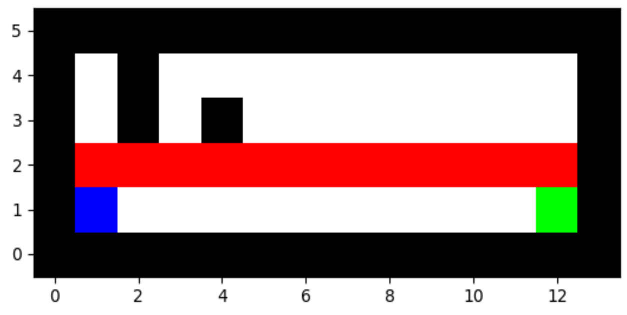

Figure 8: The optimal path towards **EXIT**

From the above figures, the differences are clear: the curves of the estimated cumulative cost are much smoother than previous ones. I think it is because in this case, the Q value functions are updated much more frequently. Hence, it allows us to use updated Q value functions during each iteration. In part 1, during each iteration(5 in total), we use the same Q value functions and only update them until all the states on the trajectory have been processed.

There is another big difference. Even though the optimal path is the same, in this case, it is less possible to find such a path. The reason is that, ϵ_k grows smaller as M grows larger. At the beginning, the Q value functions are generated randomly and we hope we could obtain better value functions after a few updates. However, there is no guarantee that the improvements are better enough for the use in the next subset with a smaller ϵ_k(We update the value functions in the first 250 iterations and fix ϵ_k as 0.4). If ϵ_k gets smaller, it means the motion model is more accurate, resulting in less exploration area. For this reason, the robot tends to select the best control action according to the current Q value functions(which may not be accurate) and ignores the other unexplored spaces.

Hence, if we could not obtain a good enough Q value functions after the first 250 iterations, with smaller ϵ_k, it is becoming less possible to find a feasible path any more. Besides, it could be worse if we keep lowering ϵ_k with longer M loops. This kind of algorithm explores large possible spaces at the beginning and then focus on smaller spaces(getting greedy). The algorithm with the same ideas is SA(Simulated Annealing).

1.4 part 3

1.4.1 formulation

Implement the Q-learning algorithm using a fixed $\epsilon_k \equiv \epsilon = 0.5$ value. Follow the template of Q-learning in the slides (page $28/32$) but make use of $M = 5$, and episodes of lengths $T_0 = 10, T_1 = 40, T_2 = 100$,$T_3 = 100$ and $T_3 = 250$, respectively in each iteration of the M loop. In the initial trajectory, take the start state to be START, then at the next iterations of the M loop, generate a new episode from the the state you reached in previous iteration (unless it is the EXIT, in which case, repeat from the START or any other state). Are the optimal paths close or far from the cliff after algorithm termination? How about the obstacle? What differences do you observe with the SARSA result from 1?

1.4.2 solution

The only difference between SARSA and Q learning is the action selection, so I only have to modify this part(Q value functions update) in the codes I used in part 1. I put the results as follow:

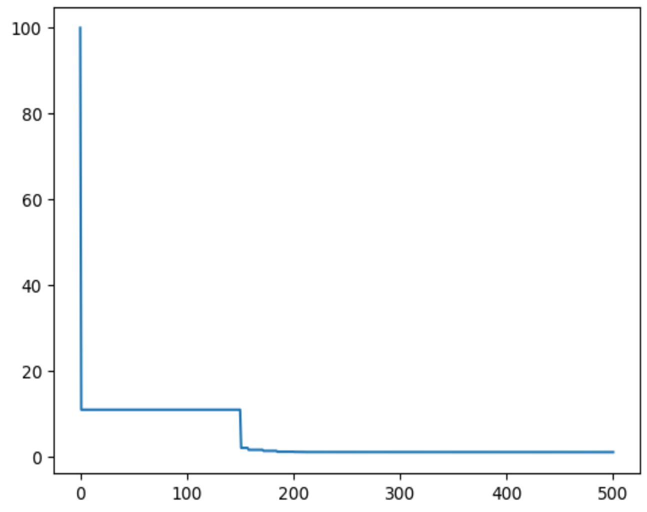

Figure 9: The estimated cumulative cost of the **START** state

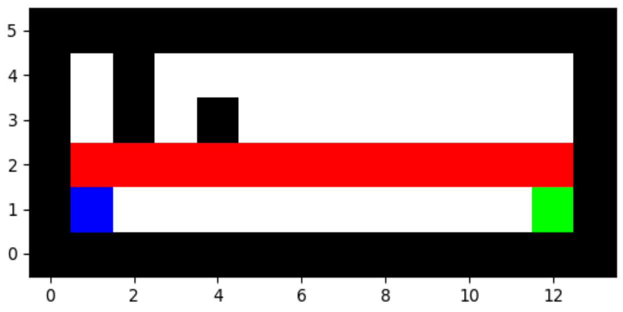

Figure 10: The optimal path towards **EXIT**

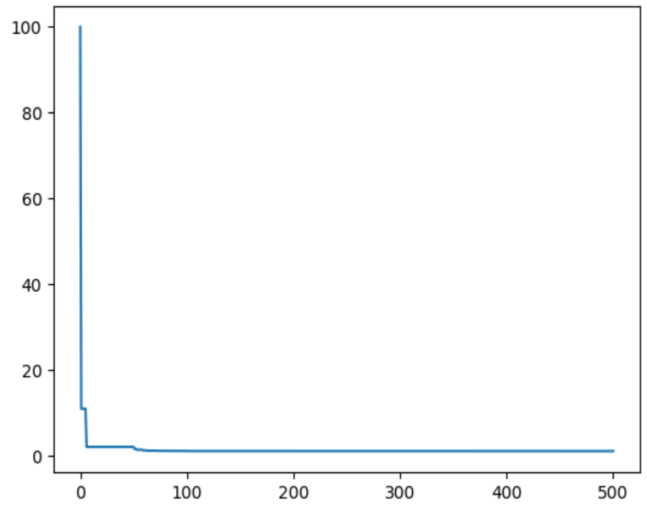

Figure 11: The estimated cumulative cost of the **START** state

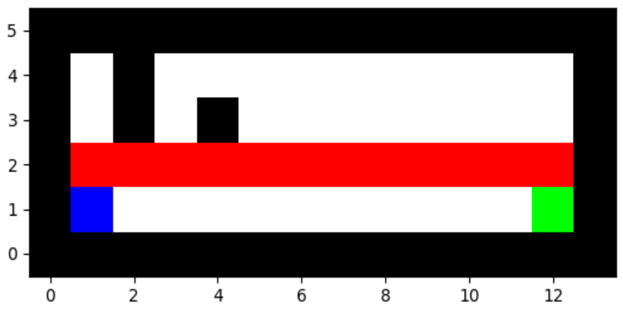

Figure 12: The optimal path towards **EXIT**

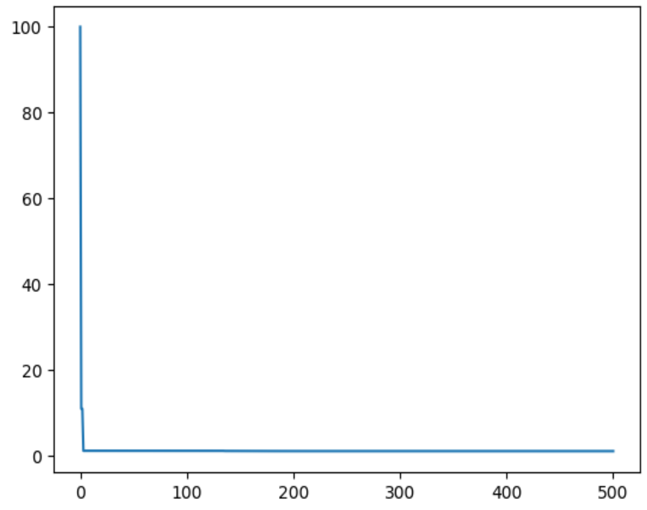

Figure 13: The estimated cumulative cost of the **START** state

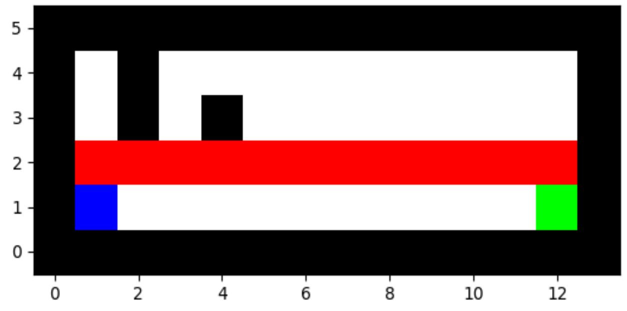

Figure 14: The optimal path towards **EXIT**

The optimal paths are the same(close to the cliff and bypass the obstacles). The differences can be found from the curves of the estimated cumulative cost. You could see that in this case, the curves are steeper and they looks like stairs. In part 1, there are also some fluctuations along the curve but they can not be detected any more.

1.5 part 4

1.5.1 formulation

Now implement the Q algorithm using k sequence of 4ϵ values: $\epsilon_0 = 0.5, \epsilon_1 = 0.3, \epsilon_2 = 0.2$ and $\epsilon_3 = 0.1$. However, instead of following the template in the slides make use of $M = 1000$ and $T = 2$ long episodes. This version of the algorithm will allow you to use more directly an updated Q function, and is the most widely used version of Q learning (see [1]). Split $M = 1000$ into sequences of 250 after which you can update from the larger to the smaller ϵ_k. Plot the value of the estimated cumulative cost from the START as time t evolves from $t = 0$ to $T = 1000$. Do you observe any differences with the previous solution? Is the optimal path any different? What happens if you keep lowering ϵ with Tonger M loops? What differences do you observe with SARSA from 3?

1.5.2 solution

The only difference between SARSA and Q learning is the action selection, so I only have to modify this part(Q value functions update) in the codes I used in part 2. I put the results as follow:

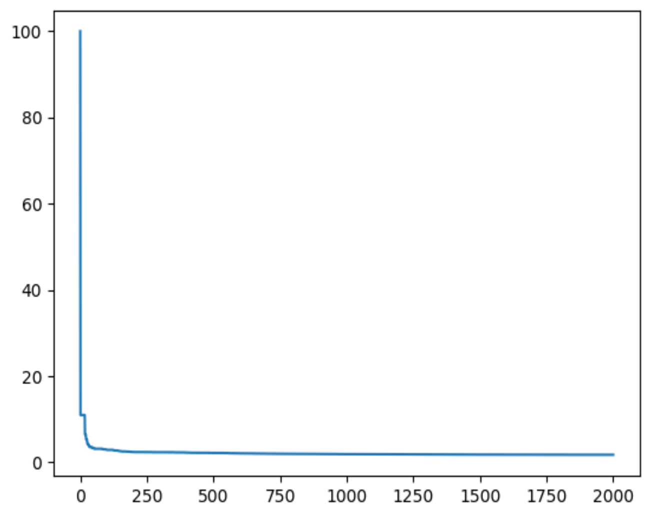

Figure 15: The estimated cumulative cost of the **START** state

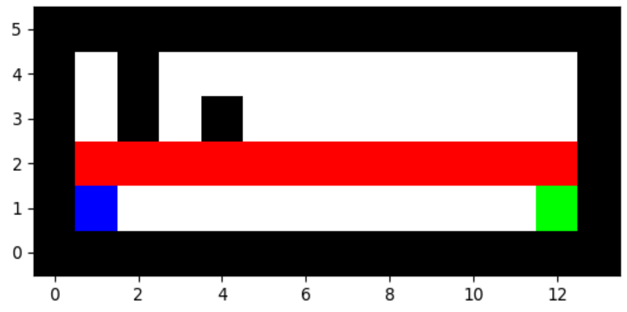

Figure 16: The optimal path towards **EXIT**

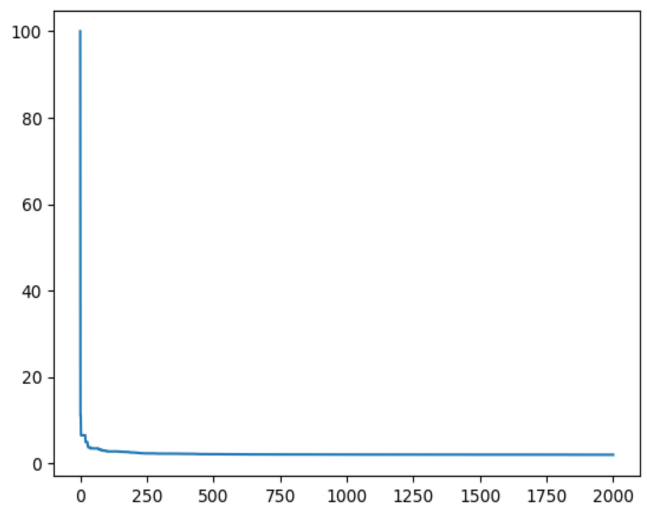

Figure 17: The estimated cumulative cost of the **START** state

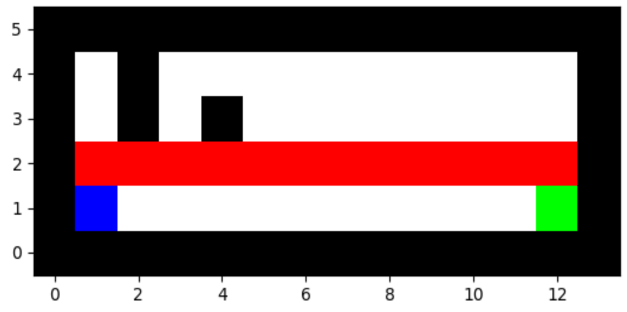

Figure 18: The optimal path towards **EXIT**

The optimal paths are the same. The main difference is that the shape of the curve is more similar to them in part 3 instead of in part 2(SARSA), which means that the curves are less smoother . if we keep lowering ϵ_k with longer M loops, it is less possible for us to find a feasible path because the algorithms would be more greedy, focusing on the current best solution.