# Assignment Three Report

Hang Zheng 520021911347

March 30, 2023

## Contents

## 1 Introduction

In this assignment, I implemented a program with two kinds of model-free control algorithms, including on-policy learning (Sarsa) and off-policy learning (Q-learning), to solve the Cliff Walking Problem under the GridWorld MDP environment without the knowledge of the MDP problem (model free).

## 2 Experimental Principle

In this section, I will give a brief introduction to the principle of the two algorithms we mentioned above.

### 2.1 $\epsilon$-Greedy Exploration

$\epsilon$-greedy exploration is a commonly used exploration strategy in reinforcement learning, which aims to balance the trade-off between exploiting current knowledge and exploring unknown states and actions with a certain probability.

In reinforcement learning, agents need to **balance exploration and exploitation.** If the agent only exploits current knowledge, it may become trapped in local optima and fail to discover better policies. Conversely, if the agent only explores, it may not exploit existing knowledge and may get stuck in infinite loops or spend a lot of time in states with no reward.

$\epsilon$-greedy exploration introduces randomness to balance exploration and exploitation. At each time step, the agent has a probability $\epsilon$ to choose a random action for exploration, and a probability of 1-$\epsilon$ to choose the optimal action for exploitation. We can set $\epsilon$ to be a fixed value, but a more common way is that as time goes on, we can typically decreases the value of $\epsilon$ gradually, so that the agent is more inclined to explore unknown states and actions in the early exploration phase, and to exploit existing knowledge in the later phase.

The formula of $\epsilon$-greedy exploration in reinforcement learning can be written like:

$$pi(a|s) = \begin{cases} \epsilon/m + 1 - \epsilon, if \ a^* = \underset{a \in A}{argmax}\, Q(s,a) \\ \epsilon/m \end{cases}$$

And we can prove that using $\epsilon$-greedy exploration will improve the value functions of the states. The proof progress is showed below:

$$\begin{aligned} q_\pi(s, \pi'(s)) &= \sum_{a \in A} \pi'(a|s) q_\pi(s,a) \\ &= \epsilon/m \sum_{a \in A} \pi'(a|s) + (1-\epsilon) \underset{a \in A}{max}\, q_\pi(s,a) \\ &\geq \epsilon/m \sum_{a \in A} \pi'(a|s) + (1-\epsilon) \sum_{a \in A} \frac{\pi(a|s) - \epsilon/m}{1-\epsilon} q_\pi(s,a) \\ &= \sum_{a \in A} \pi(a|s)\pi'(a|s) \\ &= v_\pi(s) \end{aligned}$$

Therfore, $v_{\pi'}(s) = v_\pi(s)$

In summary, $\epsilon$-greedy exploration aims to balance exploration and exploitation by introducing a certain degree of exploration, which helps to discover better policies in reinforcement learning.

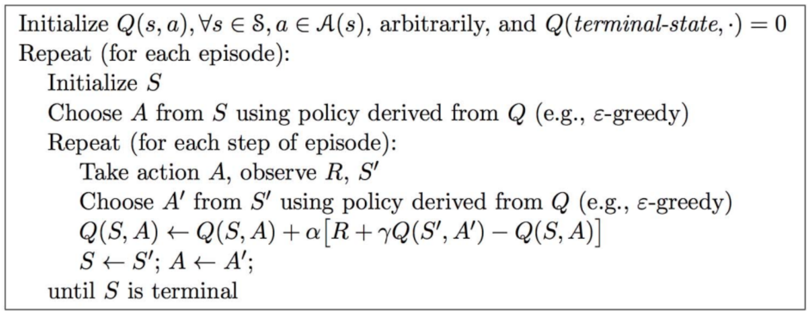
### 2.2 Sarsa Algorithm

Sarsa algorithm(State-Action-Reward-State-Action) is a on-policy algorithm that estimates the state-action value function $Q(s,a)$ to select the optimal policy. Its main idea is to combine the two processes of exploration and decision-making using an update strategy $\epsilon$-greedy exploration to continuously iterate.

Concretely, Sarsa algorithm consists of the following parts:

- Randomly initializes a state (in this case, the Cliff Walking problem, initialized the state at the Start state), uses **$\epsilon$-greedy** to select an upcoming action under the current state, and then starts the episode iteration
- Take the current action, move to the next state, and select an next action **using $\epsilon$-greedy exploration**
- Update the Q value of the current state based on Q(state,action), Q(state',action') and the reward
- Move, assign state with state' and action with action'
- Repeat the above process until reaching the terminal state

The pseudo code of the algorithm is as showed in figure 1.



```
Initialize $Q(s,a), \forall s \in \mathcal{S}, a \in \mathcal{A}(s)$, arbitrarily, and $Q(\textit{terminal-state}, \cdot) = 0$
Repeat (for each episode):
    Initialize $S$
    Choose $A$ from $S$ using policy derived from $Q$ (e.g., $\varepsilon$-greedy)
    Repeat (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
```

Figure 1: Pseudo code of Sarsa Algorithm

## 2.3 Q-Learning Algorithm

Q-Learning algorithm is a off-policy algorithm that estimates the state-action value function $Q(s,a)$ to select the optimal policy. Its main idea is to separate the processes of exploration and decision-making , using an update strategy $\epsilon$-greedy algorithm to explore while a greedy selection to improve the policies.

Concretely, Q-Learning algorithm consists of the following parts:

- Randomly initializes a state (in this case, the Cliff Walking problem, initialized the state at the Start state), then starts the episode iteration
- **Uses $\epsilon$-greedy** to select an upcoming action under the current state
- Takes the current action, move to the next state, and select an next action **using greedy selection based on the Q-values**
- Update the Q value of the current state based on Q(state,action), Q(state',action') and the reward
- Move to the next state, assign state with state'(but does not assign the action with action')
- Repeat the above process until reaching the terminal state

The pseudo code of the algorithm is as showed in figure 2.

```
Initialize $Q(s,a), \forall s \in \mathcal{S}, a \in \mathcal{A}(s)$, arbitrarily, and $Q(\textit{terminal-state}, \cdot) = 0$
Repeat (for each episode):
    Initialize $S$
    Repeat (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
```

Figure 2: Pseudo code of Q-Learning Algorithm

### 2.4 Comparison between Sarsa and Q-Learning

Sarsa and Q-learning are two widely used algorithms in reinforcement learning that address the problem of finding an optimal policy for an agent to take actions in an environment to maximize a cumulative reward. While both algorithms aim to learn the optimal policy, there are some differences between them in terms of **how they update the Q-values and handle exploration-exploitation trade-offs.**

- **Sarsa is an on-policy algorithm, while Q-learning is an off-policy algorithm**. In Sarsa, the Q-value of a state-action pair is updated based on the current policy that is being used, meaning that the next action is selected based on the same policy. On the other hand, Q-learning updates the Q-value of a state-action pair using the maximum Q-value of the next state, regardless of the action taken in the next state. This means that Q-learning is not affected by the exploration-exploitation trade-off during the learning process.
- **Sarsa is more conservative than Q-learning.** Sarsa takes into account the immediate reward and the expected future rewards when updating the Q-value, while Q-learning only considers the maximum expected future reward. This makes Sarsa more stable, as it tends to avoid large changes in Q-values, while Q-learning may be more prone to fluctuations in the learned policy.
- In terms of performance, the choice of algorithm depends on the characteristics of the problem and the environment. Sarsa is generally more suitable for problems where a cautious approach is required, and where the environment is noisy or uncertain. Q-learning, on the other hand, is more suitable for problems where exploration is less important, and where the environment is relatively predictable.

Another difference between Sarsa and Q-learning is that

## 3 Experimental Content

The directory structure of my code is like:

- main.py: The main file to be run, in which I create an instance of the MDP_Agent class I implemented to complete the task of this assignment.
- algo.py: I implement a class named MDP_Agent to provide the functionality of the two algorithms we mentioned in the last section and a extra class named GridWorld to instantiate a Gridworld environment.

Here I will list the methods of each classes and the further details will not be explained here (Please refer to the detailed comments I added in the source code)

### 3.1 GridWorld

The methods of Class GridWorld are showed as below:

- setTerminal(x,y): set the state (x,y) as a terminal state.
- setStart(x,y): set the state (x,y) as a start state.
- setCliff(x,y): set the state (x,y) as a cliff state.
- getStart(): return the start state of the GridWorld.
- isTerminal(state): return a flag to indicate whether a state is a terminal state.
- isCliff(state): return a flag to indicate whether a state is a cliff state.
- getReward(state, action, next_state): return the reward from state to next_state taking action.
- getStates(): return all the states of current mdp environment.
- getPossibleActions(state): return a list of actions that the agent able to take at current state.
- isAllowed(x,y): return a flag to indicate whether a state (x,y) is legal in current mdp environment.
- getTransition(): return the transition between states and corresponding probability.
- getRandomState(): return a random non-terminal state.

## 3.2 MDP_Agent

The methods of Class MDP_Agent are showed as below:

- runSarsa(i): run the Sarsa algorithm for i iterations. The implementation of the algorithm is basically the same as the pseudo code.
- runQLearning(i): run the Q-Learning algorithm for i iterations.The implementation of the algorithm is basically the same as the pseudo code.
- getNextAction(state, epsilon_greedy=True): Return a next action for the current state using either $\epsilon$-greedy exploration or Q-value greedy selection based on the flag epsilon_greedy.
- extractPolicy(): extract policy out of Q-values of the states.
- extractPath(): extract a best path starting from the start state and end at the terminal state that the algorithm selects based on the Q-values.
- printValuesAndPolicy(): print the values and policies of the states.

## 3.3 Program Running

To run the codes, you only need to type in the command like:

```
python main.py
```

and the program will run and save the running result in a file named "output.txt"

Further more, the program are implemented to takes several arguments and you are able to tune the hyper-parameters flexibly.

- '-i','–iteration': the minimum iterations to run the two methods.
- '-d','–discount': the discounting factor.
- '-e','–epsilon': the epsilon in $\epsilon$-greedy exploration.
- '-o','–output_file': the output file path to save the results.
- '-a','–alpha': the step size in the two algorithms when updating the Q-values.

## 4 Experimental Result

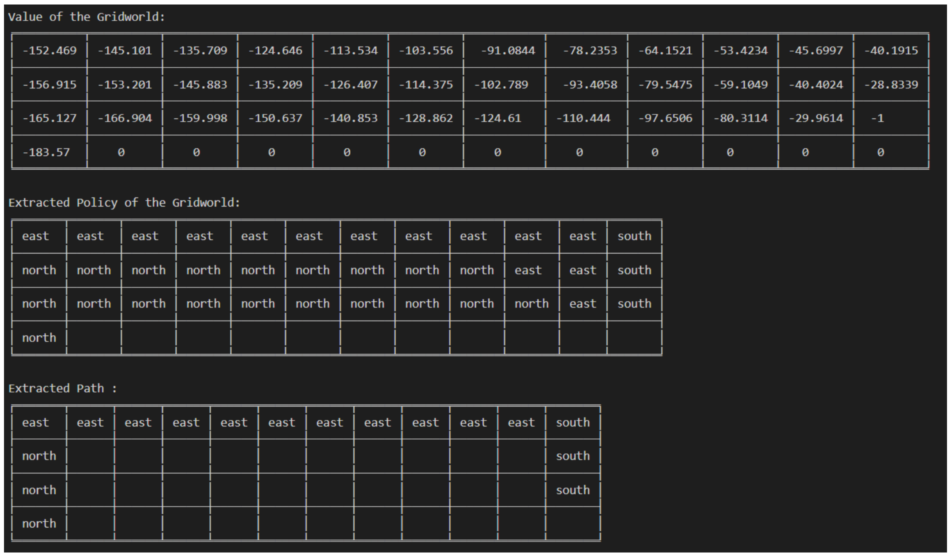
The result of the experiment is by default redirected and saved in the file ”output/output.txt”, here I will showed the result as well.

The arguments are set as: $discount\ factor:\ 1.0,\ alpha:\ 0.01,\ iteration:\ 100000$

In the experiment, I run the two algorithms with different setting of $\epsilon$ value like 0.5,0.4,0.3,0.2,0.1,0.07,0.05,0.02,0.01,0.00 within which I pick three representative results and show them in the following subsections.

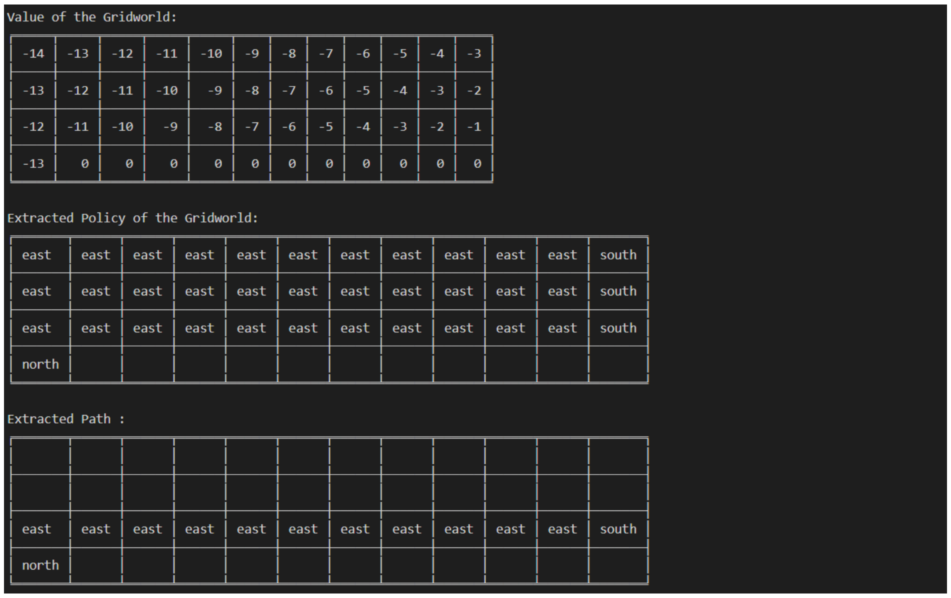
### 4.1 epsilon=0.5

The result of conducting the two algorithms with $\epsilon = 0.5$ is showed as in figure 3 and figure 4



```
Value of the Gridworld:
| -152.469 | -145.101 | -135.709 | -124.646 | -113.534 | -103.556 |  -91.0844 |  -78.2353 | -64.1521 | -53.4234 | -45.6997 | -40.1915 |
| -156.915 | -153.201 | -145.883 | -135.209 | -126.407 | -114.375 | -102.789  |  -93.4058 | -79.5475 | -59.1049 | -40.4024 | -28.8339 |
| -165.127 | -166.904 | -159.998 | -150.637 | -140.853 | -128.862 | -124.61   | -110.444  | -97.6506 | -80.3114 | -29.9614 | -1       |
| -183.57  |    0     |    0     |    0     |    0     |    0     |    0      |    0      |    0     |    0     |    0     |    0     |

Extracted Policy of the Gridworld:
| east  | east  | east  | east  | east  | east  | east  | east  | east  | east  | east  | south |
| north | north | north | north | north | north | north | north | north | east  | east  | south |
| north | north | north | north | north | north | north | north | north | north | east  | south |
| north |       |       |       |       |       |       |       |       |       |       |       |

Extracted Path :
| east  | east | east | east | east | east | east | east | east | east | east | south |
| north |      |      |      |      |      |      |      |      |      |      | south |
| north |      |      |      |      |      |      |      |      |      |      | south |
| north |      |      |      |      |      |      |      |      |      |      |       |
```

Figure 3: Result of Sarsa with $\epsilon = 0.5$

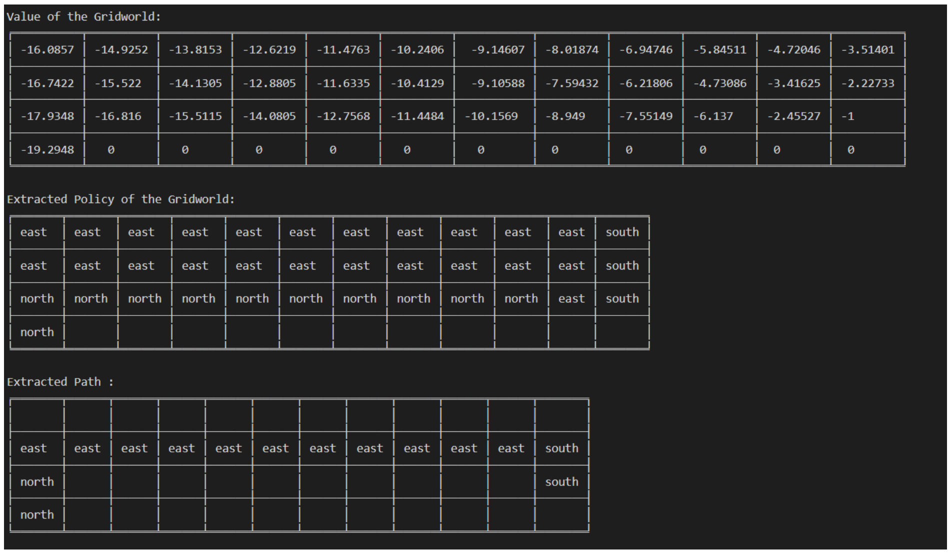


```
Value of the Gridworld:
| -14 | -13 | -12 | -11 | -10 | -9 | -8 | -7 | -6 | -5 | -4 | -3 |
| -13 | -12 | -11 | -10 |  -9 | -8 | -7 | -6 | -5 | -4 | -3 | -2 |
| -12 | -11 | -10 |  -9 |  -8 | -7 | -6 | -5 | -4 | -3 | -2 | -1 |
| -13 |   0 |   0 |   0 |   0 |  0 |  0 |  0 |  0 |  0 |  0 |  0 |

Extracted Policy of the Gridworld:
| east  | east | east | east | east | east | east | east | east | east | east | south |
| east  | east | east | east | east | east | east | east | east | east | east | south |
| east  | east | east | east | east | east | east | east | east | east | east | south |
| north |      |      |      |      |      |      |      |      |      |      |       |

Extracted Path :
|       |      |      |      |      |      |      |      |      |      |      |       |
|       |      |      |      |      |      |      |      |      |      |      |       |
| east  | east | east | east | east | east | east | east | east | east | east | south |
| north |      |      |      |      |      |      |      |      |      |      |       |
```

Figure 4: Result of Q-Learning with $\epsilon = 0.5$

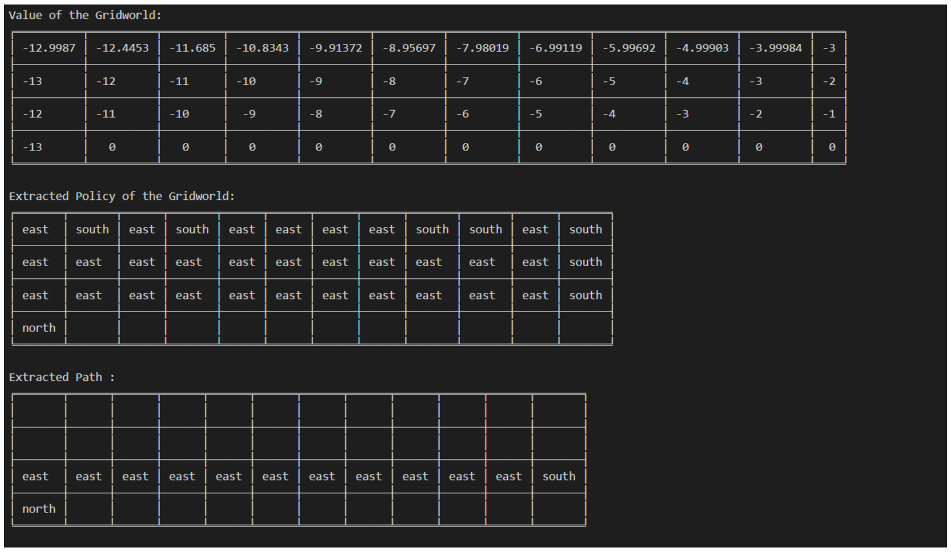
## 4.2 epsilon=0.1

The result of conducting the two algorithms with $\epsilon = 0.1$ is showed as in figure 5 and figure 6



```
Value of the Gridworld:

| -16.0857 | -14.9252 | -13.8153 | -12.6219 | -11.4763 | -10.2406 | -9.14607 | -8.01874 | -6.94746 | -5.84511 | -4.72046 | -3.51401 |
| -16.7422 | -15.522  | -14.1305 | -12.8805 | -11.6335 | -10.4129 | -9.10588 | -7.59432 | -6.21806 | -4.73086 | -3.41625 | -2.22733 |
| -17.9348 | -16.816  | -15.5115 | -14.0805 | -12.7568 | -11.4484 | -10.1569 | -8.949   | -7.55149 | -6.137   | -2.45527 | -1       |
| -19.2948 | 0        | 0        | 0        | 0        | 0        | 0        | 0        | 0        | 0        | 0        | 0        |

Extracted Policy of the Gridworld:

| east  | east  | east  | east  | east  | east  | east  | east  | east  | east  | east | south |
| east  | east  | east  | east  | east  | east  | east  | east  | east  | east  | east | south |
| north | north | north | north | north | north | north | north | north | north | east | south |
| north |       |       |       |       |       |       |       |       |       |      |       |

Extracted Path :

|       |      |      |      |      |      |      |      |      |      |      |       |
| east  | east | east | east | east | east | east | east | east | east | east | south |
| north |      |      |      |      |      |      |      |      |      |      | south |
| north |      |      |      |      |      |      |      |      |      |      |       |
```

Figure 5: Result of Sarsa with $\epsilon = 0.1$



```
Value of the Gridworld:

| -12.9987 | -12.4453 | -11.685 | -10.8343 | -9.91372 | -8.95697 | -7.98019 | -6.99119 | -5.99692 | -4.99903 | -3.99984 | -3 |
| -13      | -12      | -11     | -10      | -9       | -8       | -7       | -6       | -5       | -4       | -3       | -2 |
| -12      | -11      | -10     | -9       | -8       | -7       | -6       | -5       | -4       | -3       | -2       | -1 |
| -13      | 0        | 0       | 0        | 0        | 0        | 0        | 0        | 0        | 0        | 0        | 0  |

Extracted Policy of the Gridworld:

| east  | south | east | south | east | east | east | east | south | south | east | south |
| east  | east  | east | east  | east | east | east | east | east  | east  | east | south |
| east  | east  | east | east  | east | east | east | east | east  | east  | east | south |
| north |       |      |       |      |      |      |      |       |       |      |       |

Extracted Path :

|       |      |      |      |      |      |      |      |      |      |      |       |
|       |      |      |      |      |      |      |      |      |      |      |       |
| east  | east | east | east | east | east | east | east | east | east | east | south |
| north |      |      |      |      |      |      |      |      |      |      |       |
```

Figure 6: Result of Q-Learning with $\epsilon = 0.1$

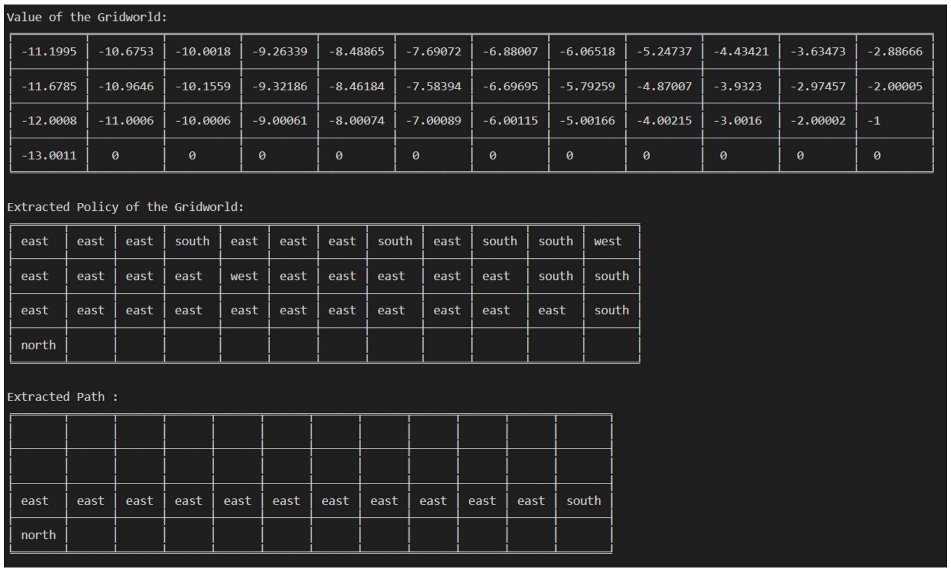
## 4.3 epsilon=0.001

The result of conducting the two algorithms with $\epsilon = 0.001$ is showed as in figure 7 and figure 8

## 4.4 Analysis

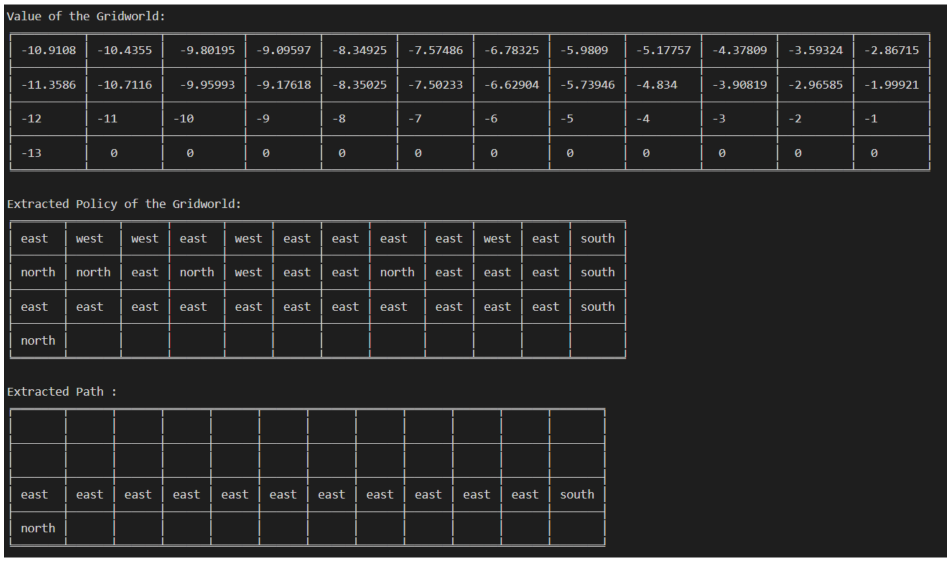
With the experimental results showed in the last several subsections, we can see that:

- With $\epsilon = 0.5$, the Sarsa algorithm chooses the most conservative path(away from the cliff but longer) while the Q-Learning algorithm chooses the shortest but most dangerous path(stay close to the cliff).



```
Value of the Gridworld:

| -11.1995 | -10.6753 | -10.0018 | -9.26339 | -8.48865 | -7.69072 | -6.88007 | -6.06518 | -5.24737 | -4.43421 | -3.63473 | -2.88666 |
| -11.6785 | -10.9646 | -10.1559 | -9.32186 | -8.46184 | -7.58394 | -6.69695 | -5.79259 | -4.87007 | -3.9323  | -2.97457 | -2.00005 |
| -12.0008 | -11.0006 | -10.0006 | -9.00061 | -8.00074 | -7.00089 | -6.00115 | -5.00166 | -4.00215 | -3.0016  | -2.00002 | -1       |
| -13.0011 |  0       |  0       | 0        | 0        | 0        | 0        | 0        | 0        | 0        | 0        | 0        |

Extracted Policy of the Gridworld:

| east  | east | east | south | east | east | east | south | east | south | south | west  |
| east  | east | east | east  | west | east | east | east  | east | east  | south | south |
| east  | east | east | east  | east | east | east | east  | east | east  | east  | south |
| north |      |      |       |      |      |      |       |      |       |       |       |

Extracted Path :

|       |      |      |      |      |      |      |      |      |      |      |       |
|       |      |      |      |      |      |      |      |      |      |      |       |
| east  | east | east | east | east | east | east | east | east | east | east | south |
| north |      |      |      |      |      |      |      |      |      |      |       |
```

Figure 7: Result of Sarsa with $\epsilon = 0.001$



```
Value of the Gridworld:

| -10.9108 | -10.4355 |  -9.80195 | -9.09597 | -8.34925 | -7.57486 | -6.78325 | -5.9809  | -5.17757 | -4.37809 | -3.59324 | -2.86715 |
| -11.3586 | -10.7116 |  -9.95993 | -9.17618 | -8.35025 | -7.50233 | -6.62904 | -5.73946 | -4.834   | -3.90819 | -2.96585 | -1.99921 |
| -12      | -11      | -10       | -9       | -8       | -7       | -6       | -5       | -4       | -3       | -2       | -1       |
| -13      |  0       |  0        |  0       |  0       |  0       |  0       |  0       |  0       |  0       |  0       |  0       |

Extracted Policy of the Gridworld:

| east  | west  | west | east  | west | east | east | east  | east | west | east | south |
| north | north | east | north | west | east | east | north | east | east | east | south |
| east  | east  | east | east  | east | east | east | east  | east | east | east | south |
| north |       |      |       |      |      |      |       |      |      |      |       |

Extracted Path :

|       |      |      |      |      |      |      |      |      |      |      |       |
|       |      |      |      |      |      |      |      |      |      |      |       |
| east  | east | east | east | east | east | east | east | east | east | east | south |
| north |      |      |      |      |      |      |      |      |      |      |       |
```

Figure 8: Result of Q-Learning with $\epsilon = 0.001$

- With $\epsilon = 0.1$, the Sarsa algorithm chooses the second conservative path(away from the cliff but longer) while the Q-Learning algorithm still chooses the shortest but most dangerous path(stay close to the cliff).
- With $\epsilon = 0.001$, the Sarsa algorithm and the Q-Learning algorithm choose the same shortest but most dangerous path(stay close to the cliff).

Through the other experiments I've conducted, I found that in Sarsa algorithm, the value of $\epsilon$ between 0.1 and 0.02 is the first turning point to make the agent choose a different path while the value of $\epsilon$ between 0.01 and 0.001 is the second one.

We can see that the Sarsa algorithm is a conservative algorithm to prefer a safer but longer(less reward) path while the Q-Learning algorithm is a radical and brave algorithm to prefer a more dangerous but shorter(more reward) path. The reason for this difference is that while predict and select the next action for the next state, the Sarsa algorithm uses $\epsilon$-greedy exploration and will finally take this action while the Q-Learning algorithm uses greedy selection based on the Q-values and are not guarantee to take this action. The greedy selection based on the Q-values makes the Q-Learning algorithm eager to take the shortest path to get the greatest reward.

Furthermore, I conducted a small experiment on the convergence of the two algorithms and find that the Sarsa algorithm converges faster and easier than the Q-Learning algorithm. The Q-values of each state in the Q-Learning algorithm fluctuate up and down and are difficult to converge while in the Sarsa algorithm, they are much more stable. This result validates the introduction and comparison we mentioned in section 2.3

## 5 Conclusion

During the process of this experiment, I first systematically learned the relevant knowledge of two basic kinds of model-free MDP control algorithms, the Sarsa algorithm and the Q-Learning algorithm, on the basis of the course content, and then implemented the two algorithms on my own to better understanding the implementation details.

Furthermore, just as in the last assignment, based on the first simple version, I further implement the code so that it can be adapted to different Gridworld problems, and even different more complex MDP problems (only if it implements specific methods such as getStates(), etc.) In the future, I may further optimized the logic of the code and test it on more complex MDP problems.