Assignment One Report

Hang Zheng 520021911347

March 16, 2023

1 Introduction

In this assignment, I implemented a program to solve the shortest path problem under the Gridworld MDP environment, which mainly consists of three parts as below:

- Policy evaluation of random policy
- Policy Iteration algorithm
- Value Iteration algorithm

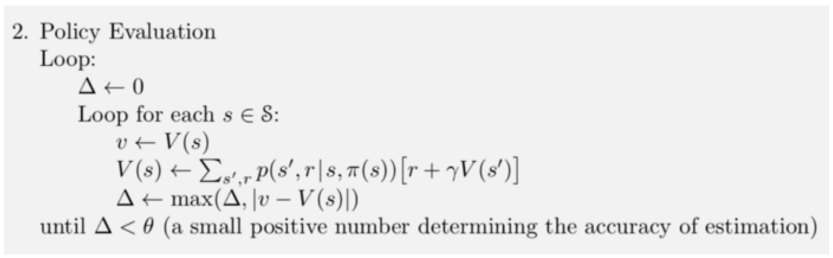
2 Experimental Principle

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

2.1 Policy evaluation

Policy Evaluation is a part of the Policy Iteration algorithm. The main idea of Policy Evaluation is: Based on the existing policy π, the algorithm traverse every state s in the Markov Decision Process (MDP), and iteratively update the value function V using the Bellman equation, until the termination condition is satisfied and convergence is reached.

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



```
2. Policy Evaluation
   Loop:
      $\Delta \leftarrow 0$
      Loop for each $s \in \mathcal{S}$:
         $v \leftarrow V(s)$
         $V(s) \leftarrow \sum_{s',r} p(s', r \mid s, \pi(s)) \big[r + \gamma V(s')\big]$
         $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
   until $\Delta < \theta$ (a small positive number determining the accuracy of estimation)
```

Figure 1: Pseudo code of Policy Evaluation

2.2 Policy Iteration

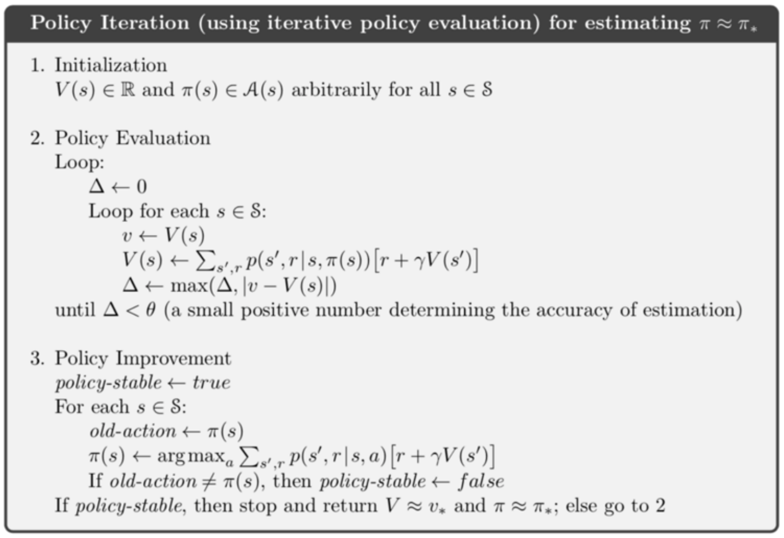
Policy Iteration is an iterative algorithm that first calculates the value function based on the current policy, then improves the policy by constructing a new one according to the value function, and repeats this process until the policy no longer changes. The algorithm converges quickly but requires solving optimization problems for both the value function and the policy at each iteration, resulting in significant computational cost.

Policy Iteration mainly consists of two part:

- Policy Evaluation, which we have discussed in the last subsection.

- Policy Improvement. Based on the existing value function V, the policy π is updated by applying the Bellman equation to each state s in the Markov Decision Process (MDP), and the policy is evaluated to determine whether it has changed. If the policy has not changed, the value function V and policy π are returned. Otherwise, go back to and repeat the process of policy evaluation.

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



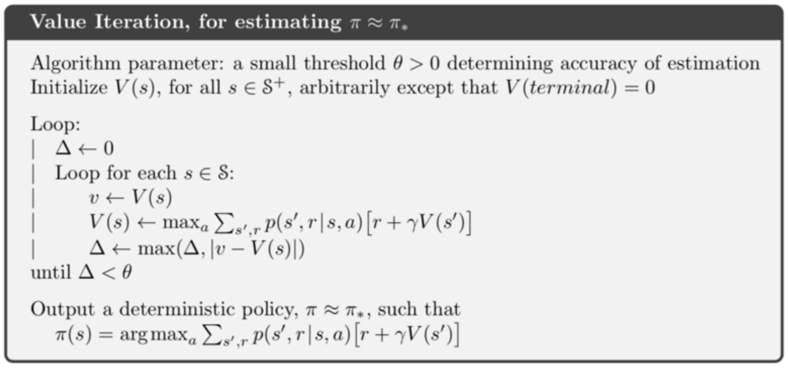
```
Policy Iteration (using iterative policy evaluation) for estimating $\pi \approx \pi_*$

1. Initialization
   $V(s) \in \mathbb{R}$ and $\pi(s) \in \mathcal{A}(s)$ arbitrarily for all $s \in \mathcal{S}$

2. Policy Evaluation
   Loop:
       $\Delta \leftarrow 0$
       Loop for each $s \in \mathcal{S}$:
           $v \leftarrow V(s)$
           $V(s) \leftarrow \sum_{s',r} p(s',r \mid s,\pi(s))\big[r + \gamma V(s')\big]$
           $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
   until $\Delta < \theta$ (a small positive number determining the accuracy of estimation)

3. Policy Improvement
   $policy\text{-}stable \leftarrow true$
   For each $s \in \mathcal{S}$:
       $old\text{-}action \leftarrow \pi(s)$
       $\pi(s) \leftarrow \arg\max_a \sum_{s',r} p(s',r \mid s,a)\big[r + \gamma V(s')\big]$
       If $old\text{-}action \neq \pi(s)$, then $policy\text{-}stable \leftarrow false$
   If $policy\text{-}stable$, then stop and return $V \approx v_*$ and $\pi \approx \pi_*$; else go to 2
```

Figure 2: Pseudo code of Policy Iteration

2.3 Value Iteration

Value Iteration is another iterative algorithm that directly starts from the initial state and, based on the Bellman Optimality Principle, gradually approximates the optimal value function through iterative updates until it converges to the optimal policy. The algorithm has the advantages of simplicity and fewer iterations but requires updating the entire state space at each iteration, resulting in significant computational cost.

The pseudo code of the algorithm is as showed in figure 3.



```
Value Iteration, for estimating $\pi \approx \pi_*$

Algorithm parameter: a small threshold $\theta > 0$ determining accuracy of estimation
Initialize $V(s)$, for all $s \in \mathcal{S}^+$, arbitrarily except that $V(terminal) = 0$

Loop:
|  $\Delta \leftarrow 0$
|  Loop for each $s \in \mathcal{S}$:
|      $v \leftarrow V(s)$
|      $V(s) \leftarrow \max_a \sum_{s',r} p(s',r \mid s,a)\big[r + \gamma V(s')\big]$
|      $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
until $\Delta < \theta$

Output a deterministic policy, $\pi \approx \pi_*$, such that
   $\pi(s) = \arg\max_a \sum_{s',r} p(s',r \mid s,a)\big[r + \gamma V(s')\big]$
```

Figure 3: Pseudo code of Value Iteration

2.4 Comparison

In summary, the Policy Iteration algorithm has a larger computational cost per iteration but converges faster, while the Value Iteration algorithm has a smaller computational cost per iteration but requires more iterations to converge. The choice of algorithm depends on the specific problem and requirements.

3 Experimental Content

The directory structure of my code is like:

- main.py: The main file to be run, in which I create several instances of the classes I implemented to complete the task of this assignment.
- model.py: I implement three classes named RandomPolicyEvaluation, PolicyAgent and ValueAgent to provide the functionality of the three 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)
- isTerminal(state)
- getReward(state, action, next_state)
- getStates()
- getPossibleActions(state)
- isAllowed(x,y)
- getTransition()

3.2 RandomPolicyEvaluation

The methods of Class RandomPolicyEvaluation are showed as below:

- runRandomPolicyEvaluation()
- extractPolicy()
- printValuesAndPolicy()
- getPolicy()

3.3 PolicyAgent

The methods of Class PolicyAgent are showed as below:

- initPolicy(policy,count,time)
- runPolicyIteration()
- computeActionFromValues(state)
- printValuesAndPolicy()

3.4 ValueAgent

The methods of Class ValueAgent are showed as below:

- runValueIteration()
- extractPolicy()
- printValuesAndPolicy()

3.5 Running

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

```
python main.py
```

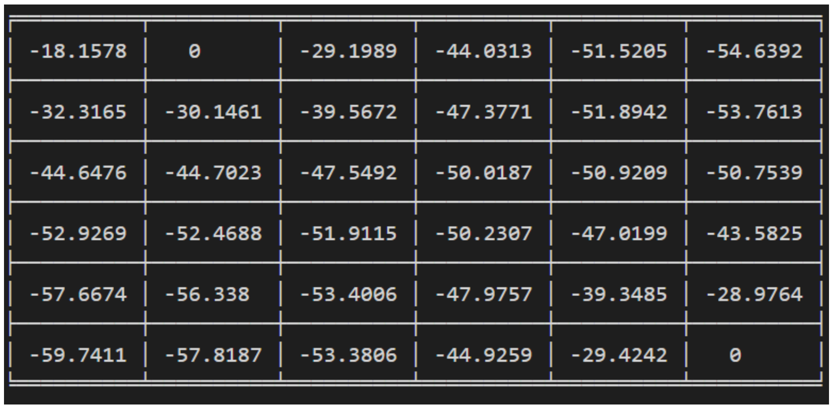
and the program will run and save the running result in a file named ”output.txt”

4 Experimental Result

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

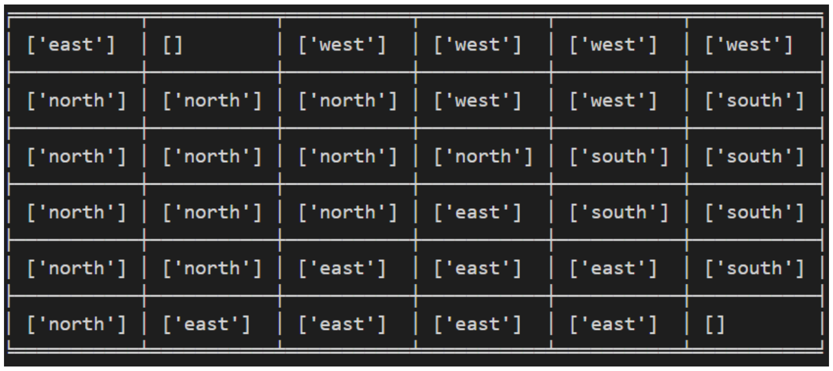
4.1 Policy Evaluation

The result of conducting policy evaluation on random policies is showed as in figure 4 and figure 5



```
| -18.1578 | 0        | -29.1989 | -44.0313 | -51.5205 | -54.6392 |
| -32.3165 | -30.1461 | -39.5672 | -47.3771 | -51.8942 | -53.7613 |
| -44.6476 | -44.7023 | -47.5492 | -50.0187 | -50.9209 | -50.7539 |
| -52.9269 | -52.4688 | -51.9115 | -50.2307 | -47.0199 | -43.5825 |
| -57.6674 | -56.338  | -53.4006 | -47.9757 | -39.3485 | -28.9764 |
| -59.7411 | -57.8187 | -53.3806 | -44.9259 | -29.4242 | 0        |
```

Figure 4: Values in Policy Evaluation Experiment



```
| ['east']  | []        | ['west']  | ['west']  | ['west']  | ['west']  |
| ['north'] | ['north'] | ['north'] | ['west']  | ['west']  | ['south'] |
| ['north'] | ['north'] | ['north'] | ['north'] | ['south'] | ['south'] |
| ['north'] | ['north'] | ['north'] | ['east']  | ['south'] | ['south'] |
| ['north'] | ['north'] | ['east']  | ['east']  | ['east']  | ['south'] |
| ['north'] | ['east']  | ['east']  | ['east']  | ['east']  | []        |
```

Figure 5: Policies in Policy Evaluation Experiment

4.2 Policy Iteration

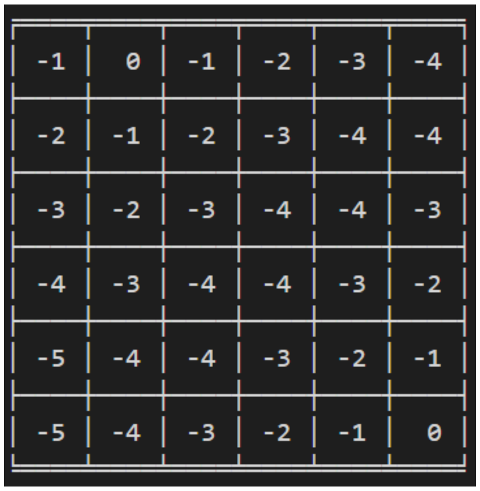
The result of Policy Iteration is showed as in figure 6 and figure 7

4.3 Value Iteration

The result of Value Iteration is showed as in figure 8 and figure 9

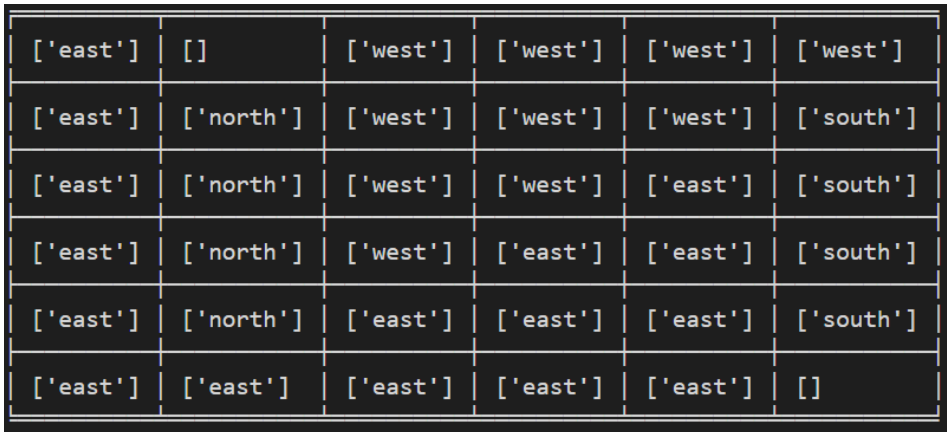
4.4 Comparison of Policy Iteration and Value Iteration

The comparison result between Policy Iteration and Value Iteration is showed as in table 1



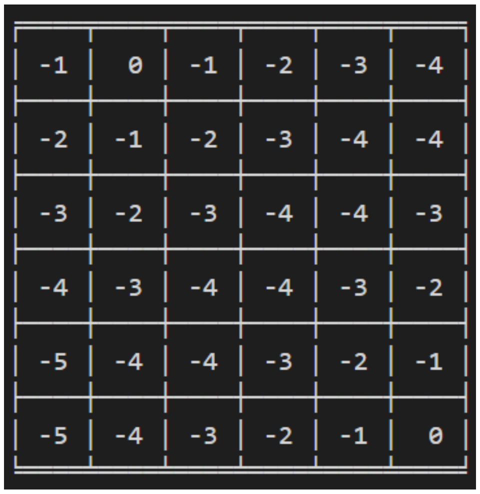
```
-1 |  0 | -1 | -2 | -3 | -4
-2 | -1 | -2 | -3 | -4 | -4
-3 | -2 | -3 | -4 | -4 | -3
-4 | -3 | -4 | -4 | -3 | -2
-5 | -4 | -4 | -3 | -2 | -1
-5 | -4 | -3 | -2 | -1 |  0
```

Figure 6: Values in Policy Iteration Experiment



```
['east'] | []        | ['west'] | ['west'] | ['west'] | ['west']
['east'] | ['north'] | ['west'] | ['west'] | ['west'] | ['south']
['east'] | ['north'] | ['west'] | ['west'] | ['east'] | ['south']
['east'] | ['north'] | ['west'] | ['east'] | ['east'] | ['south']
['east'] | ['north'] | ['east'] | ['east'] | ['east'] | ['south']
['east'] | ['east']  | ['east'] | ['east'] | ['east'] | []
```

Figure 7: Policies in Policy Iteration Experiment

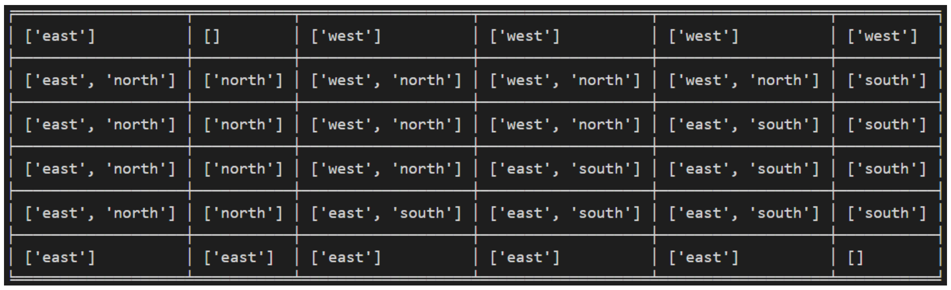


```
-1 |  0 | -1 | -2 | -3 | -4
-2 | -1 | -2 | -3 | -4 | -4
-3 | -2 | -3 | -4 | -4 | -3
-4 | -3 | -4 | -4 | -3 | -2
-5 | -4 | -4 | -3 | -2 | -1
-5 | -4 | -3 | -2 | -1 |  0
```

Figure 8: Values in Value Iteration Experiment

Algorithm	Running Time(s)	Converging Iterations	Value Function Updating
Policy Iteration	0.15533541	2	354
Value Iteration	0.00284000	6	6

Table 1: Comparison between two algorithms



```
| ['east']          | []          | ['west']          | ['west']          | ['west']          | ['west']  |
| ['east', 'north'] | ['north']   | ['west', 'north'] | ['west', 'north'] | ['west', 'north'] | ['south'] |
| ['east', 'north'] | ['north']   | ['west', 'north'] | ['west', 'north'] | ['east', 'south'] | ['south'] |
| ['east', 'north'] | ['north']   | ['west', 'north'] | ['east', 'south'] | ['east', 'south'] | ['south'] |
| ['east', 'north'] | ['north']   | ['east', 'south'] | ['east', 'south'] | ['east', 'south'] | ['south'] |
| ['east']          | ['east']    | ['east']          | ['east']          | ['east']          | []        |
```

Figure 9: Policies in Value Iteration Experiment

4.5 Analysis

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

- The results of the two algorithms are totally the same, which indicates that both of them are well and correctly implemented.
- Although the number of value function updates in policy iteration (354) is significantly higher than that in value iteration (6), the total number of iterations in policy iteration (2) is only half of that in value iteration (6), resulting in a faster convergence speed. However, due to the need to generate a policy at each iteration, the overall running speed of policy iteration is still slower than that of value iteration.

5 Conclusion

During the process of this experiment, I first systematically learned the relevant knowledge of policy iteration and value iteration on the basis of the course content, and then implemented the two algorithm on my own to better understanding the implementation details.

Furthermore, 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.