**Assignment 4: Markov Decision Processes**

Haoran Yang – ID: hyang412 – Email: [hyang412@gatech.edu](mailto:hyang412@gatech.edu)

# The algorithms used in this project:

In this project, we will explore following two model-based MDP algorithms.

1. Value iteration
2. Policy iteration

Later, we will apply 2 RL algorithms.

1. Q-learning
2. SARSA

# Task1 - MDP problem - FrozenLake

The problem background:

Winter is here. You and your friends were tossing around a frisbee at the park when you made a wild throw that left the frisbee out in the middle of the lake. The water is mostly frozen, but there are a few holes where the ice has melted. If you step into one of those holes, you'll fall into the freezing water. At this time, there's an international frisbee shortage, so it's absolutely imperative that you navigate across the lake and retrieve the disc. However, the ice is slippery, so you won't always move in the direction you intend.

The agent controls the movement of a character in a grid world. Some tiles of the grid are walkable, and others lead to the agent falling into the water. Additionally, the movement direction of the agent is uncertain and only partially depends on the chosen direction. The agent is rewarded for finding a walkable path to a goal tile.The episode ends when you reach the goal or fall in a hole. You receive a reward of 1 if you reach the goal, and zero otherwise.

Like description, this classical problem is very similar to the ‘hot beach’ one we study in the lecture. In this project, we will use gym library to build the environment, a 8\*8 grid word as FrozenLake as following.

Text

Description automatically generated

Figure 1 FrozenLake 8\*8

# Task 2 - Model-based solutions: value iteration VS policy iteration

In this section, we will apply two Model-based solutions to solve the FrozenLake MDP problem.

## Value iteration

Value iteration is a method of computing an optimal policy for an MDP and its value. In value iteration, we compute the optimal state value function by iteratively updating the estimate .

Text, letter

Description automatically generated

Figure 2 Pseudocode - value iteration

In value iteration, the converge speed depends on how we define convergence. In another word, it depends on the pre-setting parameters . We will set it differently to test.

## Policy iteration

Value iteration is a method of computing an optimal policy for an MDP. In policy iteration, we start by choosing an arbitrary policy . Then, we iteratively evaluate and improve the policy until convergence:

Text, letter

Description automatically generated

Figure 3 Pseudocode - policy iteration

## Results and insights:

Besides the final optimal policy, we are also interested in following topics:

1. How did you choose to define convergence?
2. How many iterations does it take to converge?
3. Which one converges faster? Why?
4. Do they converge to the same answer?
5. How did the number of states affect things, if at all?

### Q1 How did you choose to define convergence?

Value iteration mainly depend on the convergence of the numeric values, so its convergence is defined as the stable of the values. To be detailed, if the difference between the values updated in this iteration are very close to the values in last iteration, we think it convergence. So, the convergence should highly rely on the we set.

For policy iteration, there are two parts, policy evaluating part and policy improving. Here we only discuss the convergence based on the iterations in improving part. Different with value iteration, the results for policy iteration are discrete. Thus, it is more straightforward to define convergence. If the policy updated in this iteration is same to the one in last iteration, we think it convergence

### Q2 How many iterations does it take to converge?

The convergence speed also depends on the parameter **.** Thus, we experiment the different combination of and, and records the # of iterations.

First, we run value iteration.

Table 1 # of iterations to converge of VI

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| # iterations | = 0.8 | = 0.9 | = 0.95 | = 0.99 |
| = 1e-5 | 47 | 92 | 156 | 383 |
| = 1e-10 | 94 | 180 | 304 | 749 |
| = 1e-20 | 172 | 314 | 496 | 1125 |

Depend on different setting of these two parameters, the # of iterations to converge for this problem could range from 47 to 1125. It is intuitive that the smaller we pick, the more iterations we need to converge. Also, a bigger discount factor also costs more iterations to converge

Then we run policy iteration.

Table 2 # of iterations to converge of PI

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| # iterations | = 0.8 | = 0.9 | = 0.95 | = 0.99 |
|  | 9 | 9 | 9 | 10 |

The result shows that policy iteration is more stable with different . The # of iterations to converge for this problem could range from 9 to 10.

### Q3 Which one converges faster? Why?

The answer really depends on how we define ‘faster’. If we are talking about the # of iterations to converge, obviously, the policy iteration converges faster than value iteration. Because in each iteration, it updates much more compared to value iteration. It only cares about the policy instead of calculating the utility. Also, the space for all utilities is much bigger than the space for all policy to explore.

However, if we are talking about the time cost in seconds, value iteration could be faster than policy iteration since each iteration costs more time in policy iteration. The reason is very straightforward. The policy evaluating part in policy iteration also cost time.

### Q4 Do they converge to the same answer?

Yes, both algorithms are guaranteed to converge to an optimal policy in the end.

A picture containing text

Description automatically generated

Figure 4 optimal policy for FrozenLake8\*8

By playing the game with the optimal policy, we can reach the goal with a success rate about 86%.

### Q5 How did the number of states affect things, if at all?

We will do a test by checking a simpler version, FrozenLake 4\*4.

Table 3 # of iterations to converge of VI for FrozenLake 4\*4

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| # iterations | = 0.8 | = 0.9 | = 0.95 | = 0.99 |
| = 1e-5 | 38 | 74 | 122 | 288 |
| = 1e-10 | 79 | 158 | 262 | 620 |
| = 1e-20 | 147 | 266 | 426 | 995 |

For value iteration, if we decrease the number of states, the # of iterations do not decrease dramatically. The cost in each iteration is lower because there are less states to calculate. However, just talking the # of iterations, they are similar.

# Task3: Reinforce Learning solutions

In this section, we will apply 2 RL algorithms. For each algorithm, we will use epsilon-greedy policy ( = 0.1) to balance exploration and exploitation. Besides, we set = 0.1 and = 0.9 for both algorithms.

### Q-learning

Text, letter

Description automatically generated

Figure 5 Pseudocode Q-learning

## SARSA

Text, letter

Description automatically generated

Figure 6 Pseudocode SARSA

## Results and Conclusions

For both algorithms, after training for 10000 steps, we get Q values which provide us a success rate about 30%.

In comparison to the cases above where we knew the model, the RL algorithm perform worse here. It is reasonable since the former provides the optimal policy by leveraging the ground truth of the model.