CSC 480 – Artificial Intelligence I
2024-25 – Spring Term
Instructor: Kenny Davila Castellanos

Homework #3: Markov Decision Processes and Frozen Lakes

Objective

The first goal of this homework is to evaluate and analyze the effect of using different random policies on Markov Decision Processes (MDP). The second objective is to implement the value-iteration and policy-extraction algorithms to find the optimal policy for a given MDP.

Summary

In this work, you will have to run two experiments for solving MDPs using the gymnasium python library and the frozen lake environment. In the first experiment, we are interested in doing policy evaluation. You will try and formally evaluate multiple policies, and then you will report your findings. In the second experiment, you will have to implement the value-iteration algorithm to find the optimal values for each state in the environment. You will then use the policy extraction algorithm to generate the optimal policy, and you will have to run a formal evaluation of this policy as well.

Problem Description

Frozen lake involves crossing a frozen lake from Start(S) to Goal(G) without falling into any Holes(H) by walking over the Frozen(F) lake. The agent may not always move in the intended direction due to the slippery nature of the frozen lake. The environment is laid out as a grid, and there are two versions: 4x4 and 8x8. We use the 8x8 version.

(a) Frozen Lake visualization

(b) 4x4 version

(c) 8x8 Version

Figure 1. Frozen Lake Visualization.

This assignment is to code and run experiments with the **FrozenLake environment** implemented in Gymnasium. For more information on the environment, look at the **Gymnasium Documentation page** and their Github.

Part 1. Fixed Policies (20%)

The first step here is to **read and understand** the example code "**tutorial.py**". Make sure that you run the code and verify that it is compatible with your python environment. You need to install the gymnasium library as well as numpy if you do not already have it.

The tutorial already includes a function that runs one experiment. This function takes as input the **environment**, a fixed **policy** and a number of **episodes** (e.g. **individual simulations**), and it will **execute** the given number of episodes/simulations. Remember that this is a stochastic environment and despite using a fixed policy, each time the outcome can vary drastically. Therefore, we need to adopt a more probabilistic view of this process. The given code will compute a few important statistics:

- ***Goals***: the number of episodes, out of the given number, that our player reached the Goal state.
- ***Holes:*** the number of episodes, out of the given number, that our player got stuck in a Hole state.
- ***total_rewards:*** accumulated utility over the given number of episodes.
- ***total_goal_steps:*** total number of steps accumulated over the episodes where the character reached the goal. For example, if out of 10 episodes, the character reached the goal 3 times, one using 15 steps, the second with 10 steps, and the last with 20 steps, then this value will be 45.

In the example code, the function receives a pre-made, random policy. It is a *random policy* in the sense that the choice of **action** (left/down/right/up) for each state was pre-assigned **randomly**. The policy is also a *deterministic policy* because each state is assigned with exactly one action. However, due to the slippery nature of this environment, the desired action might result in a different outcome, making this a **stochastic** environment. You can find more details in the official **Gymnasium environment page**.

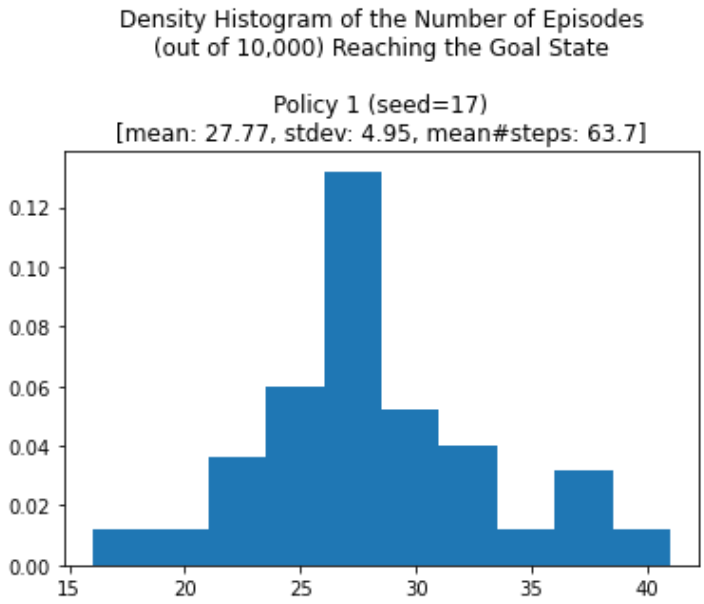
Note that the policy can be created with a fixed given seed. If you do not change the default seed value, you will always get the same policy. Your task for this part is to try several other policies (seeds) and compare their performances:

- Try **10 other policies**, using different seed numbers. For each policy,
  - Run the experiment 100 times (where each experiment runs 10,000 episodes). Note that this is basically 1,000,000 individual simulations, divided into 100 groups of 10,000 each. This will allow us to look at the variability across random groups/samples.
  - Compute and store the **mean** and the **standard deviation** (over 100 experiments) of the numbers of times the Goal was reached (**Goals**), and the average number of steps that took to reach the Goal (**total_goal_steps**).

- Select the **TOP TWO policies** based on the **mean** number of times the Goal was reached (higher is better). For **each** one of them,
    - Display the policy in a 2D array.
    - Generate a **histogram** that shows the **distribution** of the numbers of times the Goal was reached (**Distribution** of **Goals** values over **100 experiments**), with other statistics in the titles of the chart. Below is the example histogram generated for the first policy (seed=17).

```
*** Policy ***
[[2 3 0 0 1 2 3 1]
 [0 0 1 1 3 1 2 2]
 [0 2 0 0 2 1 1 2]
 [0 0 2 3 2 3 1 0]
 [2 2 2 2 1 0 2 0]
 [1 0 2 1 1 2 0 1]
 [2 3 0 3 1 1 3 0]
 [0 1 2 3 1 1 3 3]]
```

(a) Policy generated with seed=17



(b) Statistics used to evaluate this policy

Figure 2. Evaluating a random policy. Left side is the random policy. Right side is the full set of statistics expected after running 100 experiments with this policy, for a total of 10,000 episodes per experiment.

These results and their respective analysis will be included as part of your technical report.

Part 2. Optimal Policy by Value Iteration (50%)

Now we try to find the optimal policy. To that end, you will need to implement the **Value Iteration** algorithm (Sutton & Barto, 2ed, chapter 4.4). Then after the state-value function (V(s)) converges, you need to extract/derive a policy, which is the optimal policy. Your task is to implement this algorithm by yourself using the Bellman equations studied in the lectures.

The dynamics of this environment are **known already** and they are coded in the **source code of the environment**. In particular, the python dictionary '**P**' (see the tutorial.py) in the class 'FrozenLakeEnv' contains pretty much all information that you will be needing. The details of the implementation of the value iteration algorithm itself are up to you. Some of the functions shown in the tutorial.py might be really useful here. In this case, the noise is already given by the environment (2/3 chance of moving

perpendicular to target direction). There is no negative reward for the number of steps taken to reach a terminal step (cost of living = 0). Finally, use a gamma value (discount rate) = 1.0. Be sure to add a **good number of comments in the code**.

After you derive the optimal policy:

- Display the policy in a 2D array.
- Display the converged V(s) table in a 2D array.
- Generate a histogram with statistics in the title, as you did in Part 1.
- Show results and write answers to the questions shown below in the report.

The Report (30%)

You have to prepare a written report with the results for the experiments carried out here. The report should clearly state the name of the author. It must have the following sections:

1. **Introduction**
    a. Briefly explain the whole assignment in your own words.
2. **Code architecture (0.5 – 1.0 pages max)**
    a. Briefly describe how your custom code works.
    b. Usage of diagrams is highly encouraged.
    c. Code snippets might be included but you still have to explain what they do with words.
3. **Computer Specs.**
    a. Provide all the relevant specs for the computer that you used to run your experiments. In particular, mention the CPU used, the operating system, and the memory ram available.
    b. Keep in mind that a single computer should be used to collect all data.
4. **Part 1 Results.**
    a. Experimental Results:
        i. Include the top 2 policies found in your experiment.
        ii. Provide all require statistics and plots for each of these policies.
    b. An in-depth analysis of these results
        i. What do you think that makes the top 2 policies you found better than the other policies that you tried?
        ii. Is it possible to start from a random policy and gradually improve it by making changes manually? Please explain.
        iii. Anything else that you consider relevant/interesting.
5. **Part 2 Results.**
    a. Experimental Results:
        i. Include the optimal state values found by your value iteration algorithm.
        ii. Include the optimal policy found by your algorithm.

iii. Provide all require statistics and plots for the optimal policy.

b. An in-depth analysis of these results
   i. Do the actions in the optimal policy make sense to you? Please explain your answer in detail.
   ii. How much better is this policy in comparison to the best ones that you found using random actions.
   iii. Anything else that you consider relevant/interesting.

6. **Difficulties (Optional).**
   a. Provide a brief summary of difficulties that you faced while implementing this project.
7. **New application (0.25 – 0.5 pages).**
   a. Propose an entirely different application where you would use MDPs.
   b. Describe the application in detail.
   c. Be specific, what is MDP a good way to solve this problem?
8. **Feedback**
   a. What did you think about this assignment? (THIS IS NOT A conclusion)
9. **Conclusions.**
   a. NOT part of the feedback. Any final remarks and reflections on what you learned from doing this assignment.

Delivery Instructions

You are being provided with a Zip file including a basic PyCharm project. You are required to implement your solution using the code provided. When done, **you should simply zip back the entire project and submit as a ZIP file**.

1. **DO NOT rename any internal project files!**
2. **Using other compression formats (.rar) will lead to a penalty.**
3. **If you are using virtual environments in python, DO NOT include it on your zip file! It will make the project huge!**

File names. The zip file that you submit should use "[Last Name(s)], [Given Name(s)].zip" as it appears in D2L. For example, "Kenny Davila Castellanos" (Davila Castellanos is two last names), would have to submit the homework with the name "Davila Castellanos, Kenny.zip". Another student named "Kenny Mauricio Davila" (Mauricio is a middle name), would have to submit the homework as "Davila, Kenny Mauricio.zip".

Policies

1. This homework is meant to be worked **individually**.
2. All general policies about Plagiarism and Cheating apply to this mini-project. If you plagiarize or receive code from other people, you will be caught and you will receive a score of 0, and the academic integrity violation will be filed.
3. There is a data collection component in this assignment. Creating fake numbers is highly unethical and is considered Cheating.

4. You must not use Chat-GPT or any other code generators for completing this assignment. Doing so will be considered a form of cheating and are also eligible to receive a score of 0.
5. Do not post your solutions online and do not share them with anyone. It is your responsibility to safe guard your private data.
6. Code that does not compile due to syntax and/or semantic errors will automatically receive a score of 0. It is hard to assign partial credit when I cannot even run your code.
7. Must use PyCharm if you want to receive technical support from my end.
8. You must follow the delivery instructions.
9. Late submissions without justification might receive penalties proportional to the number of hours late. See the syllabus for concrete details on late penalties.
10. The code and documentation highlight using specific approaches for certain things. **You are expected to follow these approaches. Any different approach that you use without consulting me first is likely to receive a score of 0**.
11. Do ask for help if anything is unclear, but do it in a timely manner (e.g., by e-mail or during the Office Hours).