ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Institute for Dynamic Systems and Control
IDSC
Institut für Dynamische Systeme und Regelungstechnik

151-0563-01 **Dynamic Programming and Optimal Control** (Fall 2022)

Programming Exercise Topic: Infinite Horizon Problems

Issued: Nov 09, 2022 Due: Dec 18, 2022

Policy Iteration, Value Iteration, and Linear Programming

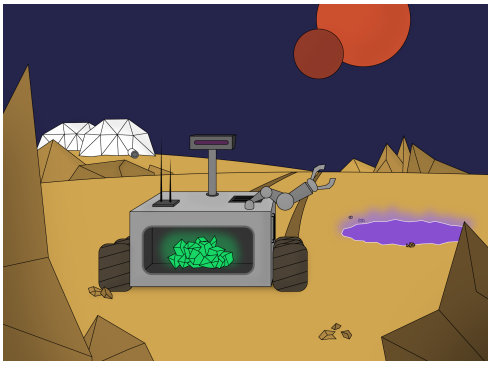

Figure 1: Your mobile robot.

You are a space explorer on a venture to new worlds. Currently, you are on a sandy planet that does not seem to offer much at first glance. However, you mapped part of the planet with a drone and detected some mysterious gemstones.

In order to examine the gems in more detail, you plan to send out your mobile robot to collect them. Due to limited fuel supplies, the mission needs to be executed as efficiently as possible. You therefore decide to use Dynamic Programming to find the optimal navigation policy for your robot.

Problem Setup

The planet has two dimensions that you call upper and lower world. The upper world consists of rocky obstacles (gray cells) and portals (P), as well as the base (B) and the laboratory (L) you have built. The lower world has obstacles and portals in the same place as the upper world, but is inhabited by hostile aliens (A). The gemstone mine (M) you have spotted is also located in the lower world. To transition between dimensions, one has to enter a portal.

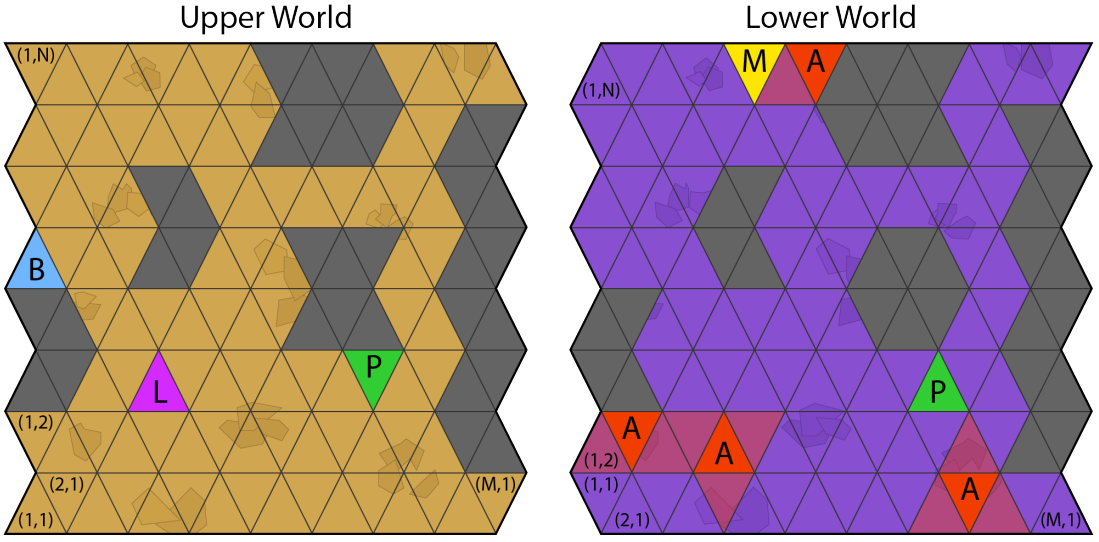

Figure 2: Top-down view of the upper and lower world. The green cell represents a portal, the blue cell the base, the purple cell the laboratory, the gray cells the rocks, the red cells the aliens and the yellow cell the mine.

Due to different laws of physics, the robot is not able to move in all four directions. The robot cannot move either north or south depending on its location. In an attempt to visualize this, the recorded map of the planet is discretized into $M \times N$ triangular shaped cells (Fig. 2), where M is the width of the world (from west to east) and N is the length (from south to north). The robot can only move to one of the adjacent triangles **(Triangular Law)**. The lower world bends space in a way, that allows the robot to move in the opposite direction of what is possible in the upper world.

The state of the robot at time step k is described by $x_k = (m, n, \phi, \psi)$, where $m \in \{1, \ldots, M\}$ and $n \in \{1, \ldots, N\}$ describe the position of the robot along the west to east and south to north axis, respectively, $\phi \in \{0, 1\}$ describes whether the robot is carrying the gems, where 1 indicates carrying and 0 indicates not carrying gems, and $\psi \in \{0, 1\}$ describes in which world the robot is, where 0 indicates the upper world and 1 indicates the lower world.

At step k, the system evolves in the following two phases:

1. One of the allowable control inputs u_k is applied. The robot is able to move **west, east** or **south/north** by one cell or can **stay** in place. If an input would violate the Triangular Law or move the robot to a cell with an obstacle or to outside the bounds of the world, that input is **not allowed**. The robot's fuel consumption of making the move at one time step is **1** (including **stay**). After the robot has made its move, the following events will take place in order:

   (a) If the robot is at one of the portals, it will enter the portal (P) and transition to the other world, i.e. from the upper world to the lower world $\psi = 0 \to 1$, or from the lower world to the upper world $\psi = 1 \to 0$.

   (b) If the robot is in the lower world $\psi = 1$, there is a probability of being attacked by the hostile aliens (A). An Alien will attack the robot no matter if the robot is carrying gems or not in the alien's cell and its adjacent cells. The robot will defend itself against an alien but needs extra fuel of $\mathbf{N_a}$. Besides that, an alien can also steal the gems if the robot is carrying gems. The probability that the robot successfully protects its gems against an alien is $p_{protected}$.

   If the robot is within the attack range of multiple aliens, the probability of protecting the gems from each alien is independent, and the fuel consumption is calculated according to the number of aliens.

   If the gems are stolen, the robot remains at its location without gems.

   (c) If the robot is at the gemstone mine (M) without carrying gems, it will collect them $\phi = 0 \to 1$.

2. After applying the input u_k, the robot **can be disturbed** to an adjacent triangle due to radiation. Depending on the dimension, the probability of being disturbed is different. In the upper world, the robot may be disturbed to **west, east or north/south** with probability $p_{disturbed}/3$. The lower world provides some shielding against the radiation, so the robot may be disturbed to **west, east or south/north** with probability $S \cdot p_{disturbed}/3$, where $S < 1$ is the shielding factor. If the robot is disturbed, it consumes extra fuel of **1**.

   In contrast to the movement of the robot in phase 1. (above), the robot may be disturbed to an obstacle cell or outside the bounds of the map. In those cases, the robot crashes immediately and loses its gems if carrying them. It needs extra fuel of $\mathbf{N_b}$ to return to the base (B) in the upper world and starts there in the next time step without gems.

   If the robot is disturbed and has not crashed at this moment, the events defined in 1.(a) - 1.(c) will take place in order again.

Note: The robot does not collide with portals (P), aliens (A), the base (B), the laboratory (L) and the gemstone mine (M) and thus can find itself in a cell with one of these.

Tasks

Find the policy minimizing the fuel consumption required to successfully collect the gems from the gemstone mine (M) and bring them to the laboratory (L) by

a) finding the index of the terminal state in the state space matrix. As you will see in `main.m /.py`, the state space matrix has K rows, where each row corresponds to a possible value that x can take.
Use the `ComputeTerminalStateIndex.m /.py` file provided as a template for your implementation.

b) creating a transition probability matrix $P \in \mathbb{R}^{K \times K \times L}$, where K is the number of possible states and L is the number of control inputs. To compute P, each entry in the state space is assigned a unique index $i = 1, 2, \ldots, K$.
Use the `ComputeTransitionProbabilities.m /.py` file provided as a template for your implementation.

c) creating a stage cost matrix $G \in \mathbb{R}^{K \times L}$.
Use the `ComputeStageCosts.m /.py` file provided as a template for your implementation.

d) **applying one of the methods: Value Iteration, Policy Iteration, Linear Programming, or a combination of these** to compute the optimal cost $J \in \mathbb{R}^K$ and the optimal policy $\mu(i),\ i = 1, \ldots, K$, that solves the stochastic shortest path problem. We are going to evaluate the solution implemented in the `Solution.m /.py` file only.

During the evaluation of your code, we will use different maps and parameters from the given ones. We will not produce unsolvable edge cases such as $M = N = 1$, or no mine and base. The correctness, efficiency and robustness of your method will be key for the evaluation. Specifically, we will evaluate the following requirements:

- Correctness of all the tasks (terminal state, transition probabilities, stage costs, solution for the SSP problem)
- Weighted score of the computational time of the implemented methods for the different tasks across different maps tested in the same computer environment. During evaluation, 4 threads will be used to run your implementation.
- In the case of a draw after the previous point calculation, the submission received first will be rewarded with more points.

Note 1: When creating the transition probability matrix, you can follow our convention of setting the probability for non allowed control inputs u to 0.
Note 2: The judgement of the TA is final.

MATLAB files provided

A set of MATLAB files is provided on the class website. Use them to solve the above problem. Follow the structure strictly as grading is automated.

`main.m`	MATLAB script that has to be used to generate the world, execute the stochastic shortest path algorithms and display the results.
`GenerateWorld.p`	MATLAB function that generates a random world.
`MakePlots.p`	MATLAB function that can plot a map of the world, and the cost and control action for each accessible cell.
`ComputeTerminalStateIndex.m`	MATLAB function template to be used to compute the index of the terminal state in the state space matrix.
`ComputeTransitionProbabilities.m`	MATLAB function template to be used for creating the transition probability matrix $P \in \mathbb{R}^{K \times K \times L}$.
`ComputeStageCosts.m`	MATLAB function template to be used for creating the stage cost matrix $G \in \mathbb{R}^{K \times L}$.
`Solution.m`	MATLAB function to implement either the value iteration, the policy iteration or the linear programming algorithm.
`exampleWorld.mat x3`	Three pre-generated worlds to be used for testing your implementations of the above functions.
`exampleP.mat x3`	The transition probability matrix $P \in \mathbb{R}^{K \times K \times L}$ for the three example worlds.
`exampleG.mat x3`	The stage cost matrix $G \in \mathbb{R}^{K \times L}$ for the three example worlds.

Note 1: Use the provided example map/P/G to test your code.
Note 2: The parameters used to get exampleP.mat and exampleG.mat are $S = 0.5$, $N_a = 3$, $N_b = 10$, $p_{disturbed} = 0.2$, $p_{protected} = 0.6$.

Python files provided

A set of Python files is provided on the class website. Use them to solve the above problem. Follow the structure strictly as grading is automated.

`main.py`	Python script that has to be used to generate the world, execute the stochastic shortest path algorithms and display the results.
`GenerateWorld.pyc`	Python function that generates a random world.
`MakePlots.pyc`	Python function that can plot a map of the world, and the cost and control action for each accessible cell.
`ComputeTerminalStateIndex.py`	Pyhton function template to be used to compute the index of the terminal state in the state space matrix.
`ComputeTransitionProbabilities.py`	Python function template to be used for creating the transition probability matrix $P \in \mathbb{R}^{K \times K \times L}$.
`ComputeStageCosts.py`	Python function template to be used for creating the stage cost matrix $G \in \mathbb{R}^{K \times L}$.
`Solution.py`	Python class to implement either the value iteration, the policy iteration or the linear programming algorithm.
`Constants.py`	Python class used to store problem constants.
`exampleWorld.mat x3`	Three pre-generated worlds to be used for testing your implementations of the above functions.
`exampleP.mat x3`	The transition probability matrix $P \in \mathbb{R}^{K \times K \times L}$ for the three example worlds.
`exampleG.mat x3`	The stage cost matrix $G \in \mathbb{R}^{K \times L}$ for the three example worlds.

Note 1: Use the provided example map/P/G to test your code.
Note 2: The parameters used to get exampleP.mat and exampleG.mat are $S = 0.5$, $N_a = 3$, $N_b = 10$, $p_{disturbed} = 0.2$, $p_{protected} = 0.6$.
Note 3: The `*.pyc` files were compiled using Python version 3.8. In order to use them, you must use Python version 3.8.
Note 4: As Python indexing starts at 0, for state $x_k = (m, n, \phi, \psi)$, $m \in \{0, \ldots, M-1\}$ and $n \in \{0, \ldots, N-1\}$. ϕ and ψ remain as described above.

Deliverables

A maximum of two students can work as one team. Please hand in by e-mail

- your MATLAB **or** Python implementation of the following files:
  `ComputeTerminalStateIndex.m /.py`,
  `ComputeTransitionProbabilites.m /.py`,
  `ComputeStageCosts.m /.py`,
  `Solution.m /.py`
- Only submit the above mentioned files.
- If you write your code in MATLAB, you are only allowed to use the basic MATLAB installation and the Parallel Computing Toolbox. The usage of any additional toolboxes is not allowed. During evaluation, MATLAB version R2022a will be used.
- If you write your code in Python, you are only allowed to make use of the Python standard library, NUMPY, SCIPY, and MATPLOTLIB. During evaluation, Python version 3.8.5, NUMPY version v1.19.2, SCIPY version v1.6.3, and MATPLOTLIB version v3.4.2 will be used.

Please include all files in one `zip`-archive, named `DPOCEx_Name1_Number1(_Name2_Number2).zip`, where `Name` is the full name of the student who worked on the solution and `Number` is the student identity number. (e.g `DPOCEx_JohnDoe_12345678(_JohnDoe_12345678).zip`)

Send your files to `dpoc.programming@gmail.com` with the subject `[programming exercise submission 2022]` by the due date indicated above. We will send a confirmation e-mail upon receiving your e-mail. You are ultimately responsible that we receive your solution in time.

Submission Checklist

- ☐ Your code runs with the original `main.m /.py` script
- ☐ You did not modify the function signatures (name and arguments) in submission files (`ComputeTerminalStateIndex.m /.py`, `ComputeTransitionProbabilites.m /.py`, `ComputeStageCosts.m /.py`, `Solution.m /.py`)
- ☐ You only submit the following files `ComputeTerminalStateIndex.m /.py`, `ComputeTransitionProbabilites.m /.py`, `ComputeStageCosts.m /.py`, `Solution.m /.py`