151-0563-01 **Dynamic Programming and Optimal Control** (Fall 2021)

Programming Exercise Topic: Infinite Horizon Problems

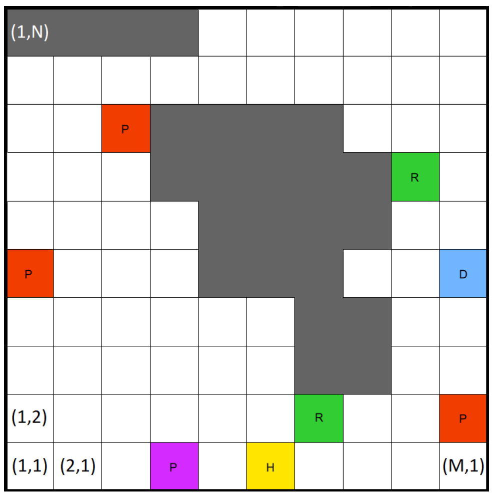
Issued: Nov 10, 2021 Due: Dec 19, 2021

Enrico Mion(enmion@ethz.ch), November 12, 2021

Policy Iteration, Value Iteration, and Linear Programming

The goal of this programming exercise is to help a bike delivery courier to deliver a pizza as fast as possible. To achieve this, the courier must go to the pizzeria to collect the pizza and then ride to a given address to deliver it. Along the way the courier must avoid situations such as policemen that slow him down and hungry residents who may steal the pizza.

(a)



(b)

Figure 1: (a) bike delivery courier. (b) Top-down view of an example world. The purple cell represents the pizzeria, the yellow cell the hospital, the blue cell the delivery station, the gray cells the buildings, the green cells the hungry residents and the red cells the police officers.

Problem Setup

The bike delivery courier operates in a world that is discretized into $M \times N$ cells (Fig. 1b), where M is the width of the world (from east to west) and N is the length (from north to south). The state of the courier at step k is described by $x_k = (m, n, \psi)$, where $m \in \{1, \ldots, M\}$ and $n \in \{1, \ldots, N\}$ describe the position of the courier along the west to east and south to north axes, respectively, and $\psi \in \{0, 1\}$, where 1 indicates that the courier is carrying a pizza and 0 indicates that he/she has no pizza.
At step k, the system evolves in the following way:

1. One of the allowable control inputs u_k is applied. The courier is able to move **north, west, south, or east** by one cell or to **take a break (stay)** in place. However, if an input would move the courier to a cell with a building or to outside the bounds of the world, that input is **not allowed**. The cost for taking an action and moving to the next cell is **one minute**, even if he/she chooses to take a break (stay). If the courier arrives in a cell with a policeman, he/she has to reduce the speed to 1/3, so the cost will be **three minutes**.

2. After applying the input, the bike delivery courier **can be distracted** with probability $p_{distracted}$. He/she may go either **north, west, south, or east, with equal probability** $p_{distracted}/4$. If the courier is in a cell with a building or leaves the bounds of the world, the bike delivery courier will get injured and be sent to the **hospital**.
   There could be an additional cost to the one described in 1) if one of the following situations takes place:

   - If the courier is injured, the cost for being sent to the hospital and recovering is $\boldsymbol{N_c}$ **minutes**, and the courier starts from the hospital at the next step **without a pizza**.
   - If the courier is distracted but not injured, the additional cost for moving to the next cell is **one minute**.
   - If the courier is distracted but not injured, and he/she arrives in a cell with a policeman, the additional cost is **three minutes**.
   - If the courier is not distracted, there will not be any additional cost.

3. If the courier is not injured, the hungry residents will attempt to steal the pizza. The probability of the pizza being stolen by a hungry resident i is

$$p_{r_i} = \begin{cases} \gamma/(d_i+1) & \text{if } 0 \leq d_i \leq R \\ 0 & \text{otherwise} \end{cases},$$

   where d_i is the L_1 distance (measured in number of cells) from the courier to the resident i, and R is the maximal searching range of the residents. **All residents within the maximal range will try to steal the pizza independently**. If the pizza is stolen, the courier must go back to the pizzeria to pick up a new pizza. After applying the action and possibly being distracted, the probability of the pizza being stolen is **decreased by a factor of 2** if the courier is in a cell with a policeman.

4. If the bike delivery courier has not been injured and not robbed by this point and is at the pizzeria carrying no pizza, he/she collects one. If the courier has not been injured by this point and is at the destination carrying a pizza, the task terminates.

Note 1: The events take place in this exact order, 1 to 4.
Note 2: The courier is fully aware of structure of the map and knows the locations of residents, policemen, hospital and pizzeria, and thus tries to make the best move at each step.
Note 3: The courier does not collide with residents, policemen, pizzeria, hospital and thus can find himself/herself in a cell with one of these. Residents can see the courier through buildings.

Tasks

Find the policy minimizing the expected time (in minutes) required to successfully deliver a pizza by

a) finding the index of the terminal state in the state space matrix. As you will see in `main.m`, the state space matrix has K rows, where each row corresponds to a possible value that x can take.
Use the `ComputeTerminalStateIndex.m` 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` 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` 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` 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 pick-up and delivery stations. 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.
- In particular, we will reward the ability to choose the best methods in the Solution.m function in terms of computation efficiency and the robustness of the solution. By robustness we mean that the solution must be able to produce reasonable results in every possible test scenario chosen by the TAs.
- In the case of a draw after the previous point calculation, the submission received first will be rewarded with more points.

Note 1: Use the provided example map/P/G to test your code. One of the maps can be used to test the robustness of the implemented method.
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: The parameters used to get exampleP.mat and exampleG.mat are $\gamma = 0.4$, $R = 5$, $N_c = 10$, $p_{distracted} = 0.1$.

Deliverables

Please hand in by e-mail

- your MATLAB implementation of the following files:
  `ComputeTerminalStateIndex.m`,
  `ComputeTransitionProbabilites.m`,
  `ComputeStageCosts.m`,
  `Solution.m`
- Only submit the above mentioned files. Your code should not depend on any other non-standard MATLAB functions.

Please include all files in one `zip`-archive, named `DPOCEx_Name_Number.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_AuthierJules_12345678.zip`)

Send your files to `enmion@ethz.ch` with the subject `[programming exercise submission]` 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 script
- ☐ You did not modify the function signatures (name and arguments) in submission files (`ComputeTerminalStateIndex.m`, `ComputeTransitionProbabilites.m`, `ComputeStageCosts.m`, `Solution.m`)
- ☐ You only submit the following files `ComputeTerminalStateIndex.m`, `ComputeTransitionProbabilites.m`, `ComputeStageCosts.m`, `Solution.m`