RL - Reinforcement Learning

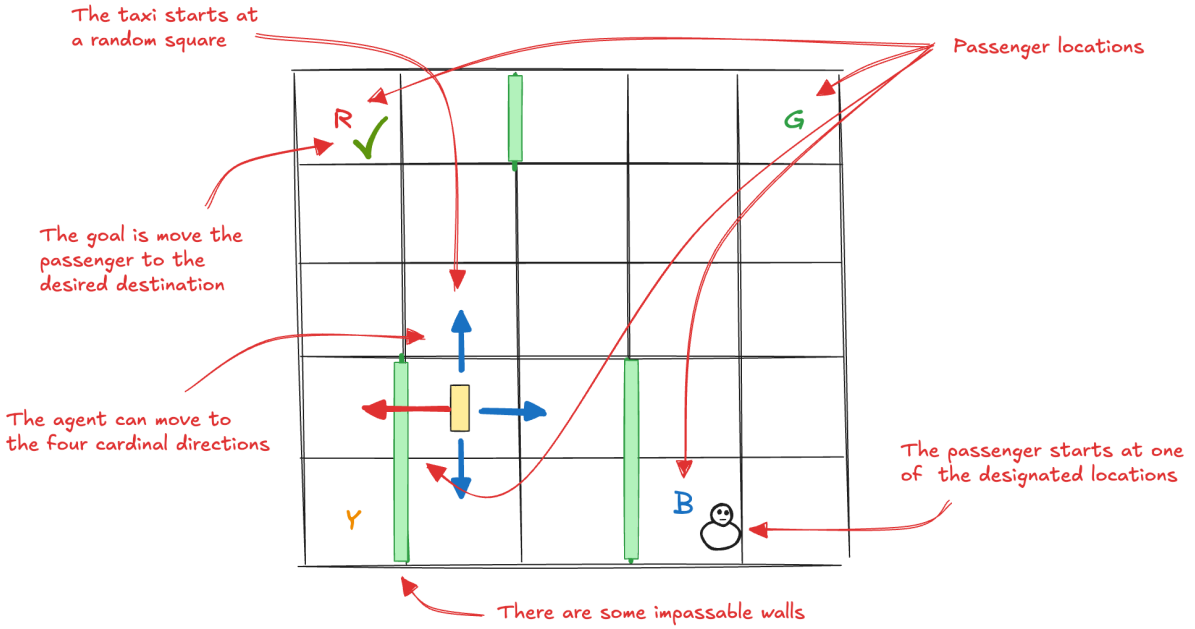
Exercises on Dynamic Programming

Prof. Riccardo Berta

2025.10.03

1 Taxi

The Taxi problem involves navigating passengers in a grid world, picking them up and dropping them off at one of four locations (Red, Green, Yellow and Blue) in a 5x5 grid world. The taxi starts off at a random square and the passenger at one of the designated locations. The goal is move the taxi to the passenger's location, pick up the passenger, move to the passenger's desired destination, and drop off the passenger. Once the passenger is dropped off, the episode ends.



There is six possible actions. The agent can move to the four cardinal directions (south, north, east or west) and two other possible actions which are the pick-up or the drop-off:

- 0: Move south (down)
- 1: Move north (up)
- 2: Move east (right)
- 3: Move west (left)
- 4: Pickup passenger
- 5: Drop off passenger

The state is described by the taxi location on the grid (a row and a column number between 0 and 4), a location to drop-off the passenger from four choices, and the passenger which can be in one of the four locations or inside the taxi, in total 500 discrete states. An observation is returned as an int that encodes the corresponding state, calculated by: ((taxi_row * 5 + taxi_col) * 5 + passenger_location) * 4 + destination

The player receives positive rewards for successfully dropping-off the passenger at the correct

location. Negative rewards for incorrect attempts to pick-up/drop-off passenger and for each step where another reward is not received.

1 - Import the 'Taxi-v3' grid word from Gymnasium:

```
# You can get the environment from Gymnasium in the same way we got 'Frozen Lake';
# in order to visually plot the environment you can import it
# using render_mode="rgb_array"
```

2- Create a random policy as a baseline:

```
# You have to create a function that get in input a state and provide a random action
# (in the range [0;5])
```

3 - Show the policy in action by rendering the environment several times after different decisions from the random policy:

```
# You have to use a while loop in order to provide the current state to
# the policy and then make a step in the environment using the action
# provided by the policy.
# Try to create a function (to be called "show_policy")
# to be reused with other policies later.
# Hints: env.render() provides you an array representing
# an image of the environment; plt.imshow() can be used to visualize
# the image on the screen.
```

2 - Write a brute-force function in order to evaluate the probability of success and the average return obtained by a policy, the evaluate the random policy:

```
# You can reuse the "evaluate" function that we already apply to
# the Frozen Lake environment, however pay attention on how to determine
# the end of an episode (it is no more state 15).
# Hints: you have the "done" information from step function.
```

3 - Use the value-iteration (or the policy-iteration) algorithm to calculate the optimal policy and also the optimal state-value function:

```
# You can reuse the "value_iteration" function or
# the "policy_iteration" function that we already
# apply to the Frozen Lake environment.
```

4 - Calculate the performance of the obtained optimal policy using the brute force approach:

```
# You can reuse the "evaluation" function written before,
# in order to evaluate the optimal policy
```

5 - Show the optimal policy:

```
# You can reuse the "show_policy" function written before,
# in order to show the optimal policy
```