Reinforcement Learning: From Foundations to Deep Approaches

Summer Semester 2025, Homework 1

Prof. Georgia Chalvatzaki, Dr. Davide Tateo

TECHNISCHE UNIVERSITÄT DARMSTADT

Use the MushroomRL Python library to solve the problems. Have a look at the practical sessions on how to use MushroomRL. It is up to you whether you want to solve the exercises on Google Colab in a jupyter notebook or locally on your PC.

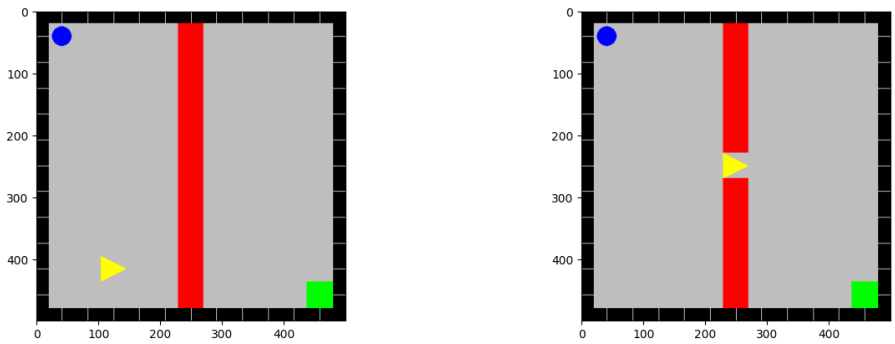
This assignment will not be graded! The solutions will be discussed on 13. June 2025 in the practical session.

1.1 The Stairway to Heaven Environment

An agent starts in the environment, aiming to reach heaven. A lava river separates our agent from heaven; whenever the agent hits the lava, it gets a penalty. The agent can build a stairway over the lava river by hitting a button in the top-left of the map. Figure 1 depicts the environment.

The agent receives a high reward of 1.0 for reaching the goal, a penalty of -0.1 for hitting the lava (The lava is like a wall, the agent gets a penalty for hitting it but stays in the same state), and a small negative reward of -0.01 for all other transitions.

Our agent can move in four directions: right, up, left, and down. There is a 5% probability of action failure (the agent stays in its position). The stairway can collapse with a 3% probability at every time step (the collapse is independent of the action transition). The stairway cannot collapse when the agent is on it.



(a) Stairway not built (b) Stairway built

Abb. 1: Stairway to heaven environment: The agent is a yellow arrowhead, and heaven is the green square. There is a lava river (red squares) that separates the agent from heaven. Hitting the button (blue circle) builds a stairway over the lava. The position of the button and the heaven are fixed. The agent is spawned randomly in the part that contains the button, and the stairway is not built initially.

The grid space gets doubled because of the event button press. The agent starts in the first grid and transfers to the next grid upon landing on the button. The button state exists only in the second grid (the state for the button in the first grid (state [0,0,0]) does not exist in the cell list; when the agent moves to the button, it jumps directly to the button position in the second grid (state[1,0,0])).

Use the MushroomRL library to solve the problem.

1. Define the Stairway to Heaven environment using the FiniteMDP class in Mushroom. Start from the following code snippet, define functions to acquire the probability transition matrix, the reward, and the initial state probability distribution, and then call the constructor of the parent class. A function to render the MDP can also be useful to test your environment.
Follow the same steps shown in Practical session 3.

```
class StairwayToHeaven(FiniteMDP):
    def __init__(self, gamma=0.99, horizon=100, a_success_prob=0.95, collapse_prob=0.03):
        # The grid: 'B' is the button, 'G' is the goal,
        #           '#' is lava, 'S' is a potential starting position
        grid_map1=np.array(
           [['B', 'S', 'S', 'S', 'S', '#', '.', '.', '.', '.', '.'],
            ['S', 'S', 'S', 'S', 'S', '#', '.', '.', '.', '.', '.'],
            ['S', 'S', 'S', 'S', 'S', '#', '.', '.', '.', '.', '.'],
            ['S', 'S', 'S', 'S', 'S', '#', '.', '.', '.', '.', '.'],
            ['S', 'S', 'S', 'S', 'S', '#', '.', '.', '.', '.', '.'],
            ['S', 'S', 'S', 'S', 'S', '#', '.', '.', '.', '.', '.'],
            ['S', 'S', 'S', 'S', 'S', '#', '.', '.', '.', '.', '.'],
            ['S', 'S', 'S', 'S', 'S', '#', '.', '.', '.', '.', '.'],
            ['S', 'S', 'S', 'S', 'S', '#', '.', '.', '.', '.', '.'],
            ['S', 'S', 'S', 'S', 'S', '#', '.', '.', '.', '.', '.'],
            ['S', 'S', 'S', 'S', 'S', '#', '.', '.', '.', '.', 'G']
            ]
        )
        H,W = grid_map1.shape
        grid_map2=grid_map1.copy()
        grid_map2[0][0]='.'
        grid_map2[np.where(grid_map2=='S')]='.'
        grid_map2[H//2][W//2]='.'
        grid_map = np.array([grid_map1, grid_map2])
        # Directions correspond to actions: Right, Down, Left, Up
        self.directions = [[0, 1], [1, 0], [0, -1], [-1, 0]]
        # Fill here ...
        super().__init__(p, r, mu, gamma, horizon)
```

1.2 Dynamic programming on Stairway to Heaven

1. Solve the problem with Policy Iteration (PI) and Value Iteration (VI). Visualize the value function and the optimal policy for PI and VI. Adapt the visualization functions from Practical session 3 to use the grid map and the cell list of the MDP. Use the visualization function for all of the following questions.
Compare the results. What do you observe?

```
def visualize_value_matrices(V, grid_map, cell_list, label):
    """
    Args:
      V (np.ndarray): value matrix with a size of [n_states]
      grid_map (np.ndarray): the MDP grid map, useful for visualization
      cell_list (list): the MDP cell list
      label (str): the title of the figure
    """
    # A similar prototype for visualize_policy() function
```

2. **Modify the Policy Iteration method in Mushroom (see Practical session 3) so that it can accept a policy initialization and stopping criterion (in terms of a number of iterations). Initialize with a policy that always goes to the RIGHT (action = 0) and compare the obtained Value Function and Policy for 3 and 10 iterations. What do you observe?**

```
def policy_iteration_modified(p, r, gamma, initial_policy=None, n_iterations=-1):
    """
    Args: ...
        initial_policy (int  or np.ndarray): value to intialize the policy.
                        if np.ndarray is provided, use it as the initial policy.
        n_iterations (int): number of iterations to run the algorithm.
                    if -1 is provided, the stopping criterion is the convergence.
    Returns:
        V, pi: The optimal value of each state and the optimal policy.
    """
```

3. **Implement the exact policy evaluation using the matrix-based formulation of the Bellman equation. Visualize the value function for the policy after 10 iterations from q.2.2.**

```
def exact_policy_evaluation(policy, p, r, gamma):
  """
  Args:
    policy (np.ndarray or a MushroomRL policy): the policy to evaluation
    ...
  """
```

1.3 Model-free RL on Stairway to Heaven

1. **Take the policy of iteration 10 of the modified policy iteration algorithm from q2.2. Implement the First-Visit Monte Carlo method and n-step TD prediction (TD(n)) for value prediction. Visualize the predicted value function for MC and TD(5). Discuss the values when the stairway is built. Why are there many zero values?**

```
def TD_policy_eval(n, policy, mdp, num_rollout=100, alpha=0.1, gamma=0.99):
    ....

def MC_policy_eval(policy, mdp, num_rollout=100, gamma=0.99):
    ....
```

2. **Train the same MDP with Q-Learning, SARSA and SARSA(λ) with an ϵ-Greedy policy. Run 5 random seeds and plot the evaluation curves for all algorithms when you evaluate the policy every n step. Plot the evaluation curves average returns J w.r.t. epochs. (see Practical session 5). Save the agent of each seed to use it later.**

   Hint: All of the agents converge after 100 epochs of 100 episodes.

3. **Visualize the predicted value matrices for the first seed for each agent from q.3.2. using MC and TD(5) policy evaluation function developed in q.3.1.**

```
def load_and_visualize(file_name):
    ....
```