Lab 2: MDP Modeling and Dynamic Programming for Routing in Communication Networks

Sindri Magnússon

Overview

In this lab, you will solve a **Stochastic Shortest Path (SSP)** problem motivated by a wireless communication scenario. You are designing a routing strategy for delivering packets in a wireless mesh network with uncertain, node-specific link behavior. The goal is to compute an optimal policy using **dynamic programming**.

Scenario: Routing in a Wireless Grid Network

You are tasked with routing a packet across a 5×5 wireless grid network. Each grid cell represents a router, and the packet must travel from a source node to a designated destination node (typically the bottom-right corner). However, due to wireless interference and local variability, routing is uncertain. We illustrate the network in Figure 1.

States: Each state represents the current location of the packet and is labeled by its grid coordinates (i, j), where i is the row index and j is the column index. For example:

- $(0, 0)$ is the top-left router,
- $(4, 4)$ is the bottom-right router (goal),
- $(2, 3)$ is a router in the middle-right region.

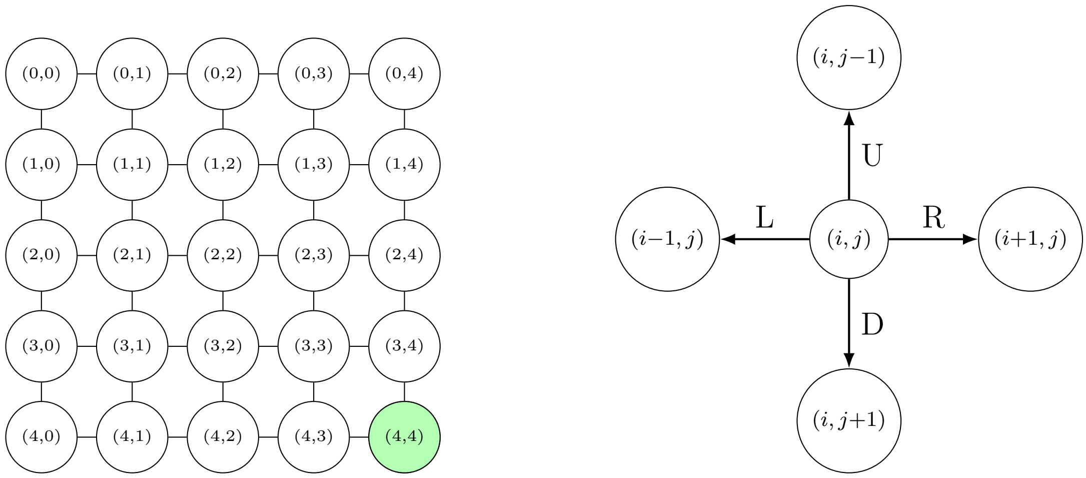
Actions: At each step, you can choose one of four possible actions to send the packet to a neighboring router:

- `'U'` — attempt to send the packet **up** (to router $(i-1, j)$),
- `'D'` — attempt to send the packet **down** (to router $(i+1, j)$),
- `'L'` — attempt to send the packet **left** (to router $(i, j-1)$),
- `'R'` — attempt to send the packet **right** (to router $(i, j+1)$).

If an action would lead the packet outside the grid (e.g., sending 'U' from row 0), the packet remains in the same state.

Transitions: Because of wireless noise and interference:

- Each node has its own stochastic transition probabilities.



(a) Full 5×5 grid with routers and edges (b) Actions at router (i, j)

Figure 1: Topology and action semantics in the wireless routing environment. (a) The 5×5 grid layout of routers, with each node labeled by its coordinates and connected to its adjacent neighbors. The goal node $(4, 4)$ is highlighted in green. (b) Interpretation of actions from a node (i, j): each action (U, D, L, R) corresponds to forwarding the packet to a neighboring router in the respective direction.

- Most of the time, the packet moves in the intended direction.
- Occasionally, it slips to a neighboring router (perpendicular to intended direction), or stays in place.
- The probabilities vary across the grid (e.g., some routers may be more congested or unreliable).

Each move incurs a negative reward of -1. Reaching the goal yields a reward of 0 and terminates the episode. Your goal is to learn a policy that minimizes the expected cumulative cost.

The implementation of the environment is provided in **Appendix 1**.

Task 1: Identify the MDP

Use the provided code to create an instance of the wireless routing environment on the 5×5 grid.

(a) Identify the MDP. A key step in working with a **Markov Decision Process (MDP)** is recognizing its components: *states*, *actions*, *transition probabilities*, and *rewards/costs*. Be prepared to briefly *describe* what each of these means for this MDP (in our wireless routing setting): what is the state and actions, how uncertainty defines $P(s' \mid s, a)$, and what per-step cost/reward is used.

(b) Explore the transitions. For a given state and action, the transition probabilities define how likely the packet is to move to each possible next state. Plot these probabilities for a specific state–action pair. For example, examine state $(2, 2)$ with action `'U'`. Try other combinations (e.g., $(0, 3)$ with `'R'`) and compare.

(c) Interpret the results. Be prepared to explain in your own words what the plotted transition probabilities mean in this context: i) Which next states are most likely? ii) What does a high probability of "staying in place" indicate? iii) How do these transitions relate to network reliability or congestion?

Task 2: Evaluate Fixed Policies via Linear System

In this task, you will evaluate the value function $V(s)$ under two fixed policies by solving a linear system of equations. Recall that the value function $V^\pi(s)$ represents the expected cumulative reward obtained by starting in state s and following policy π thereafter, and is defined as:

$$V^\pi(s) = \mathbb{E}_\pi \left[\sum_{t=0}^{\infty} R(s_t, \pi(s_t)) \,\middle|\, s_0 = s \right].$$

Policies:

- **Policy 1:** Always try to send the packet to the `'R'` (right).
- **Policy 2:** Always try to send the packet to the `'D'` (down).

In this task we compute the value function $V^\pi(s)$ for each policy by solving the linear system:

$$V^\pi = R^\pi + P^\pi V^\pi \quad \Rightarrow \quad (I - P^\pi)V^\pi = R^\pi.$$

Note. As discussed in Lecture 4, since the discount factor is $\gamma = 1$, the Bellman equations admit multiple solutions unless a boundary condition is imposed. Because the goal router $(4, 4)$ is absorbing, setting $V(4, 4) = 0$ and solving on the transient states yields a *unique* solution *provided the policy is proper*, i.e., it reaches the goal with probability 1 from every state. This holds for policies that eventually drift toward the goal (e.g., "always Down" or "always Right" when slip dynamics allow progress). In contrast, an *improper* policy (e.g., "always Up" or "always Left" on this grid) can fail to reach the goal with positive probability, making the expected cost infinite and the linear system singular.

Goal. Policy evaluation (with $\gamma = 1$ and $V(\text{goal}) = 0$) is already implemented in the notebook. Your task is to *run* the code for the two fixed policies (always `Up` and always `Down`), *inspect* the resulting value-function heatmaps V^π, and *interpret* what they show.

Be prepared to explain:

a) **Meaning of $V^\pi(s)$ and the heatmap.** For a specific state (e.g., $s = (2, 2)$), explain what the heatmap value represents under a given policy.

b) **Method.** Describe how solving a linear system is used to compute V^π when $\gamma = 1$ and $V(4, 4) = 0$ is imposed. Explain why a boundary condition is required in this setting. Briefly discuss what changes if $\gamma < 1$.

c) **Comparison.** Using the two heatmaps, state which policy appears better and justify using the pattern of values (no calculations required).

d) **Dynamic Programming alternative.** Outline how to compute V^π via DP (using *rewards* R) instead of a direct solve: write the policy-evaluation Bellman equations

$$V^\pi(s) = R^\pi(s) + \sum_{s'} P^\pi(s,s')\, V^\pi(s'), \qquad V^\pi(4,4) = 0,$$

where $R^\pi(s) = \mathbb{E}[\, R(s,\pi(s),s')\,]$. Then iterate over transient states:

$$V_{k+1}(s) \leftarrow R^\pi(s) + \sum_{s'} P^\pi(s,s')\, V_k(s'),$$

until convergence (e.g., $\|V_{k+1} - V_k\|_\infty$ below a small tolerance). [Optional: Implement the value evlatuion.]

Task 3: Simulating Policies & Estimating Value

In this Task you should use the provided code to do the following:

1. **Play with simulations.** Use the provided code to simulate trajectories under different policies (e.g., always `R`, always `D`, or a mix such as 70% `R`/30% `D`). **Optional:** You can visualize example trajectories, we provide code for this.
2. **Main experiment.** From start state $(0,0)$, run many episodes for **Policy 1** (always `R`) and **Policy 2** (always `D`). For each policy, record:
   - average return (mean over episodes),
   - mean number of steps,
   - success rate (fraction reaching the goal).
3. **Compare to Task 2.** Compare each policy's empirical *average return* to the value you computed in Task 2 (linear-system evaluation) for the same start state.

Be ready to show your work and explain:

- What is a trajectory? Why is it (typically) different at each episode? What controls the randomness?
- How the empirical average return estimates $V^\pi(s_0)$: this is *Monte Carlo* policy evaluation (this an example of a reinforcement learning algorithm, no model knowledge used, only observed states and rewards). With enough episodes, the average return converges to the true value (see Lecture 5).

Task 4: Value Iteration (Optimal Value & Policy)

Do. Run the provided function to compute the optimal value and extract the optimal policy:

$$V^*,\ \pi^* \leftarrow \texttt{value\_iteration()}.$$

Inspect the heatmap of V^* and the action map for π^*.

Be ready to explain:

- **What "optimal" means.** In your own words, state what it means for a policy/-value to be optimal in this setting (do not compute—explain the concept).

- **Idea of value iteration.** Explain that we repeatedly apply the Bellman *optimality* backup on all non-terminal states (with $g = (4, 4)$ fixed at $V(g) = 0$):

$$V_{k+1}(s) \;\leftarrow\; \max_{a\in\{U,D,L,R\}} \sum_{s'} P(s' \mid s, a) \Big(R(s, a, s') + \gamma\, V_k(s')\Big), \quad s \neq g,$$

  keeping $V_{k+1}(g) = 0$, and stop when the change is small (e.g., $\|V_{k+1} - V_k\|_\infty < \varepsilon$).

- **From value to policy.** Describe how to compute the optimal policy from V^* via one-step lookahead:

$$\pi^*(s) \;\in\; \arg\max_{a\in\{U,D,L,R\}} \sum_{s'} P(s' \mid s, a) \Big(R(s, a, s') + \gamma\, V^*(s')\Big), \quad s \neq g,$$

  and set $\pi^*(g)$ arbitrarily (unused at the absorbing goal).

Optional checks.

- Compare $V^*(s_0)$ to the values you computed in **Task 2** for the fixed policies at the same start state s_0 (the optimal value should be no worse).

- Simulate a few trajectories under π^* and qualitatively relate the observed returns/paths to the structure of V^*.