Part 2 - Project report

Årne, Runar and Lars

February 16, 2024

1 Introduction

In this project we will be using dynamic programming to compute different policies for finding the optimal path down from a mountain. We will mainly be exploring policy iteration, truncated policy iteration and value iteration. To accomplish this, we also need to implement policy evaluation and policy improvement.

2 Results

We first describe the given problem formally as an MDP (Markcov Decision Process). An MDP is described as $\langle \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, \gamma \rangle$.

The finite set of states $\mathcal{S}$ is equal to every possible position in the gridworld.

The finite set of actions $\mathcal{A}$ is composed of the directions *forward*, *upforward* and *downforward*.

As the robot's actions are deterministic, the transition probability matrix $\mathcal{T}$ is composed of all 1s.

The reward function $\mathcal{R}$ is equal to the negative roughness index of the gridworld.

The discount factor γ is arbitrary, but lets set it to 0.5.

Considering the policy π_{str} of walking straight forward, we've ran policy evaluation from left-to-right and right-to-left.

The resulting evaluations are very similar, as we can see from the plots on figure 1. However, they are not exactly equal (as can be confirmed by checking that the values within the evaluation are not all equal). They differ because we are using results from the current iteration on the right-to-left approach, but results from the last iteration when doing the left-to-right approach. Therefore they will converge slightly differently. However, as they both converge towards a common target they will be very similar.

Now we might ask the question if we learned something different between the two evaluations. Even though they differ, they are similar enough that they both predict the optimal starting row should be row 10. As they also look similar enough in the plots, we can conclude that they have learned the same thing.

Let's now construct a policy π_{rnd}, having a uniform probability between all possible actions given every state. After policy evaluation (figure 2) we see that this policy also predicts the optimal starting row to be 10. Therefore we can not say we learned anything different. We also see from the plots that they look very similar.

To compare the different policies and evaluate which is the best, we compare the evaluation from each policy and

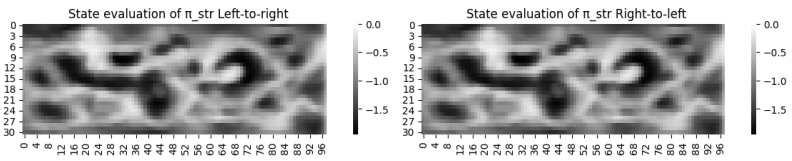

Figure 1: Heatmap of policy evaluation of π_{str} from left-to-right and right-to-left.

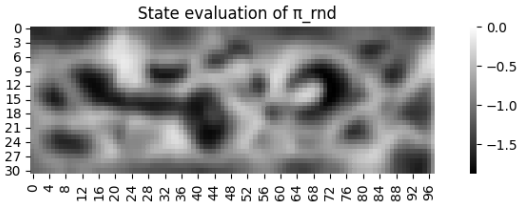

Figure 2: Heatmap of policy evaluation from of π_{rnd}.

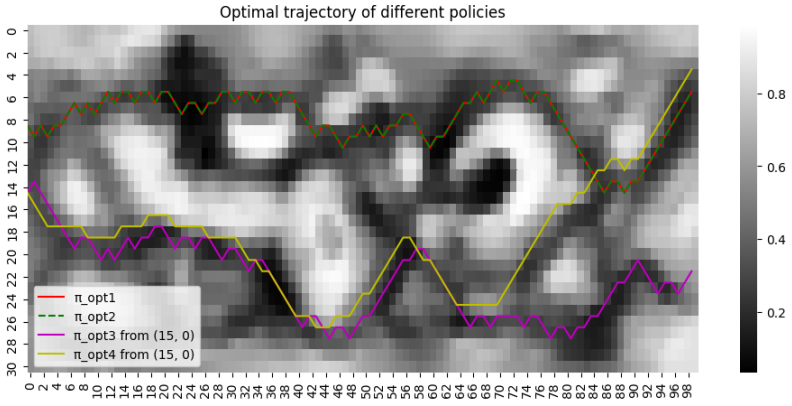

Figure 3: Heatmap of policy evaluation from of π_{rnd}.

each starting positions. From this we can conclude that if the robot could choose its initial position and policy, the optimal choice would be π_{rnd} with starting row 15. This is because $\pi_{\text{rnd}}((15,0)) \approx -0.7831$ which is slightly higher than the other $\pi_{\text{str}}((15,0)) \approx -0.8362$ and $\pi_{\text{str}}((3,0)) \approx -1.3877$.

Instead of evaluating the possible policies we are going to learn the optimal through improving π_{rnd}. After training π_{opt1} with standard policy iteration, π_{opt2} with truncated policy iteration with k=3, π_{opt3} with value iteration (through truncated policy iteration with k=1) and π_{opt4} with value iteration as a direct implementation of the Bellman optimality equation.

As we can see from figure 3, π_{opt1} and π_{opt2} are identical! π_{opt3} and π_{opt4} are similar, but they diverge after column 70.

π_{opt1} and π_{opt2} are identical as they both converge towards the same policy, only π_{opt2} takes "shorter" steps each time.

π_{opt3} and π_{opt4} are similar, but calculation of π_{opt4} "skips" the policy improvement steps, therefore "shortcutting" part of the algorithm. This will lead to faster convergence, but probably sacrifices the precision one has when calculating π_{opt3}.

The most efficient solution for policy iteration was definitely value iteration. More specifically, value iteration as a direct implementation of the Bellman optimality equation. This makes sense, as this is the policy calculation which performs the least operations (in a sense).

During the policy evaluatuion step, the states were calculated from left-to-right, meaning from column 0 to column 100.