Linear Programming Assignment

Sandro Paradžik
University of Sarajevo

December 23, 2024

1 Introduction

This document presents a solution to a Markov Decision Process (MDP) problem using linear programming (LP). The problem, as proposed by Sutton and Barto [1], involves optimizing the actions of a robot that collects soda cans while managing its battery levels efficiently.

2 Problem Statement

The robot operates in two battery states: **high** and **low**. Depending on the state, the robot can choose among several actions:

- **Search for cans**: Yields an expected reward of 2, but in the high state, it risks transitioning to the low state with probability $1 - \alpha$. In the low state, searching risks running out of battery, penalized by -3 (after this battery is set to high state).
- **Wait for cans**: Provides an expected reward of 1 and keeps the battery state unchanged.
- **Charge the battery**: Available only in the low state, it transitions to the high state without a direct reward.

The objective is to maximize the cumulative discounted reward with a discount factor $\gamma = 0.9$.

3 Mathematical Formulation

Using the Bellman optimality equations [1], the problem is formulated as an LP [2]. Let $v(h)$ and $v(l)$ represent the value functions for the high and low battery states, respectively. The rewards are defined as $r_{search} = 2$ and $r_{wait} = 1$. The constraints are derived as follows:

$$\begin{aligned}
\text{High state } (\mathbf{h})\text{:} \quad & v(h) \geq r_{wait} + \gamma v(h), \\
& v(h) \geq r_{search} + \gamma \left(\alpha v(h) + (1-\alpha) v(l)\right). \\
\text{Low state } (\mathbf{l})\text{:} \quad & v(l) \geq r_{wait} + \gamma v(l), \\
& v(l) \geq \gamma v(h), \\
& v(l) \geq \beta r_{search} - 3(1-\beta) + \gamma \left((1-\beta) v(h) + \beta v(l)\right).
\end{aligned}$$

The LP formulation is:

$$\begin{aligned}\text{Minimize:}\quad & v(h)\\ \text{Subject to:}\quad & \text{the above constraints.}\end{aligned}$$

4 Python Implementation

The Python implementation uses the `cvxpy` library to solve the linear programming formulation of the recycling robot problem. This library simplifies the creation and solving of optimization problems with constraints. Below is the key idea behind the implementation:

- The `v_h` and `v_l` variables represent the value functions for the high and low battery states, respectively.
- The constraints are derived from the Bellman equations for each state, ensuring the policy satisfies the optimality conditions.
- The objective is to minimize `v_h`, which represents the value of the high battery state.
- The implementation includes logic to determine the optimal policy ($\pi_*(h)$ and $\pi_*(l)$) based on the calculated optimal value functions.

```python
def recycling_robot(alpha, beta, r_s=2, r_w=1, gamma=0.9):
    # Decision variables
    v_h = cp.Variable(name="v_h")  # Value for high state
    v_l = cp.Variable(name="v_l")  # Value for low state

    # Objective
    objective = cp.Minimize(v_h) # we can also use v_h + v_l

    # Constraints (Bellman)
    constraints = [
        # high -> wait
        v_h >= r_w + gamma*v_h,

        # high -> search
        v_h >= r_s + gamma*(alpha*v_h + (1 - alpha)*v_l),

        # low -> wait
        v_l >= r_w + gamma*v_l,

        # low -> recharge
        v_l >= gamma*v_h,

        # low -> search
```

```
        v_l >= beta*r_s - 3*(1 - beta) + gamma*((1 - beta)*v_h + beta*v_l)
    ]

    # Solve the problem using a linear programming solver
    prob = cp.Problem(objective, constraints)
    prob.solve(solver=cp.GLPK)  # Use GLPK, an LP solver

    # Convert v_h and v_l to float
    v_h = float(v_h.value)
    v_l = float(v_l.value)

    # Calculate optimal policies
    pi_h = -1
    pi_l = -1

    eps = 0.001

    if abs(v_h - (r_w + gamma*v_h)) < eps:
      pi_h = 1 # wait
    elif abs( v_h - (r_s + gamma*(alpha*v_h + (1 - alpha)*v_l))) < eps:
      pi_h = 2 # search

    if abs( v_l - (r_w + gamma*v_l)) < eps:
      pi_l = 1 # wait
    elif abs(v_l - (gamma*v_h)) < eps:
      pi_l = 0 # recharge
    elif abs(v_l - (beta*r_s - 3*(1 - beta) + gamma*((1 - beta)*v_h + beta*v_l))) < eps:
      pi_l = 2 # search

    return {
        "v_h": v_h,
        "v_l": v_l,
        "pi_h": pi_h,
        "pi_l": pi_l
    }
```

Listing 1: Solving the problem as an LP in Python.

5 Results and Discussion

We presented an approach to solving the recycling robot problem using Python. Figure 1 illustrates the optimal value functions and policies for $\alpha \in (0, 1)$ and $\beta \in (0, 1)$, demonstrating how MDPs can be effectively solved using LP.

Another aspect of interest is the time complexity of this approach. While larger instances of MDPs are not typically solved using LP, the approach we used has strong theoretical guarantees. Specifically, LP can solve MDPs in polynomial time with respect to $|S| \cdot |A|$, where S represents the set of possible states and A the set of possible actions.

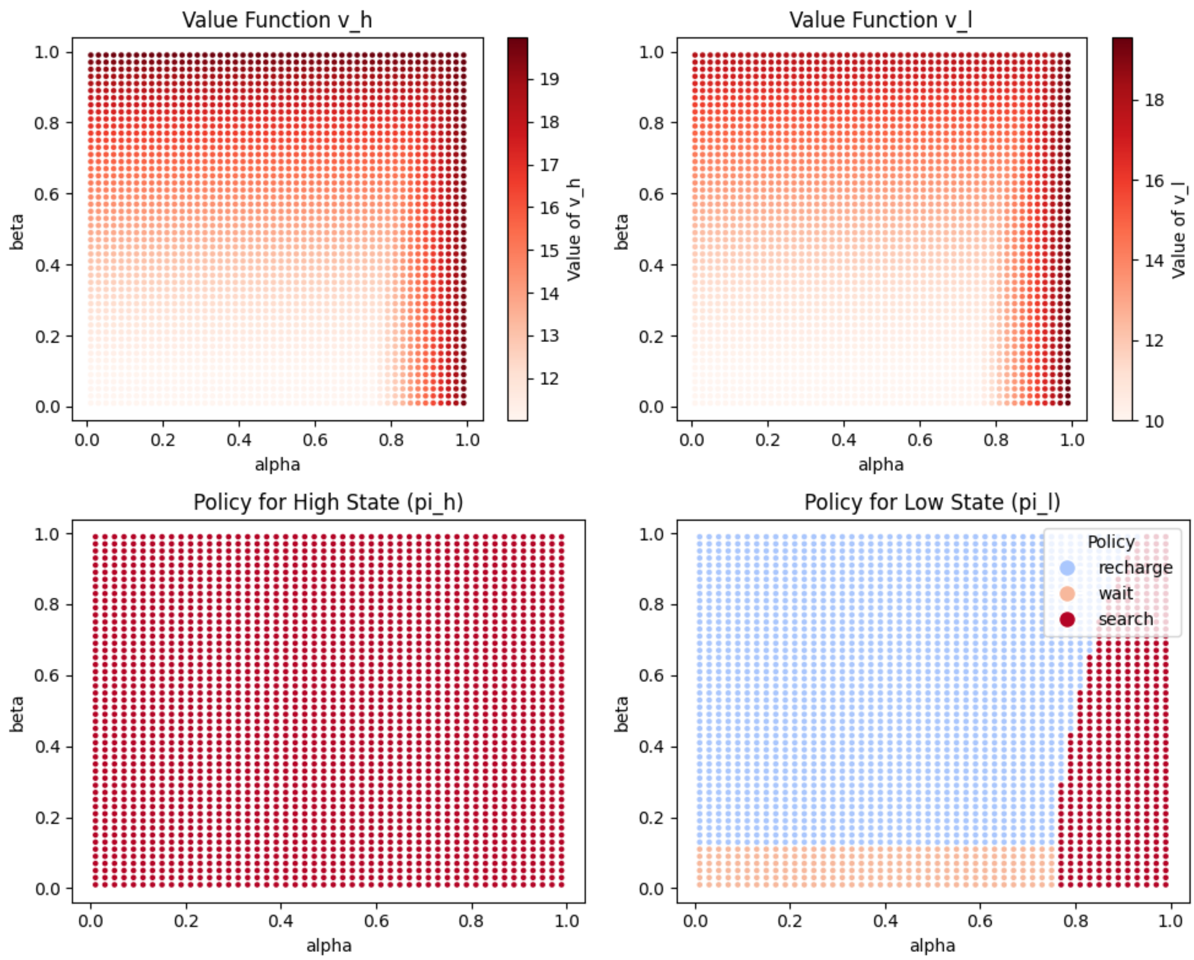

Figure 1: Results of the linear programming solution.

References

[1] Sutton, R. S., & Barto, A. G. (2018). *Reinforcement learning: An introduction* (2nd ed.). MIT Press.

[2] Helmert, M., & Röger, G. (2021). *Planning and Optimization: F2. Bellman Equation & Linear Programming*. Retrieved from `https://ai.dmi.unibas.ch/_files/teaching/hs21/po/slides/po-f02.pdf`