Value Iteration for a 5x5 Grid World Markov Decision Process

Emanda Hailu
GSR 5056/17

January 29, 2026

1 Overview

This report presents a detailed implementation of the value iteration algorithm for a fully observable Markov Decision Process (MDP) represented as a 5x5 grid world. The agent can move in four directions (up, down, left, right), receives a reward of -1 for every non-terminal step, and a reward of $+10$ for reaching the goal state. The discount factor is $\gamma = 0.9$. We describe the MDP formulation, derive the value iteration update rule, and provide Python code to compute the optimal value function and the corresponding optimal policy for each state.

2 Introduction

Reinforcement Learning (RL) studies how an agent can learn to act optimally by interacting with an environment modeled as a Markov Decision Process (MDP).[1] In an MDP, the agent observes states, selects actions, collects rewards, and transitions to new states according to a transition model.[1] Dynamic programming methods, such as value iteration, can compute optimal decision policies when the MDP model is known and the state and action spaces are finite.[2]

Grid world is a simple yet illustrative environment commonly used to demonstrate MDP concepts and RL algorithms.[1] In this report, we use a 5x5 grid world with deterministic transitions to show how value iteration converges to an optimal policy.

3 MDP Formulation

An MDP is defined by the tuple $(\mathcal{S}, \mathcal{A}, P, R, \gamma)$.[2]

3.1 State Space

The state space $\mathcal{S}$ consists of all grid cells in a 5x5 grid.[1] We index states by their row and column positions:

$$\mathcal{S} = \{(i, j) \mid i \in \{0, \ldots, 4\},\ j \in \{0, \ldots, 4\}\}.$$

We designate one state as the terminal goal state:

$$s_G = (4, 4).$$

3.2 Action Space

At each non-terminal state, the agent can choose from four deterministic actions:[1]

$$\mathcal{A} = \{\text{up, down, left, right}\}.$$

We encode them as indices:

- 0: up
- 1: down v
- 2: left <
- 3: right >

3.3 Transition Dynamics

The environment is deterministic.[1] If the agent chooses an action that would move it outside the grid boundary, it remains in the same state. Formally, let $\delta(a)$ be the movement associated with action a:

$$\delta(\text{up}) = (-1, 0), \quad \delta(\text{down}) = (1, 0), \quad \delta(\text{left}) = (0, -1), \quad \delta(\text{right}) = (0, 1).$$

For a non-terminal state $s = (i, j)$ and action a, the next state $s' = (i', j')$ is

$$(i', j') = \begin{cases} (i + \delta_i(a),\ j + \delta_j(a)) & \text{if } 0 \le i' \le 4,\ 0 \le j' \le 4, \\ (i, j) & \text{otherwise.} \end{cases}$$

Thus, the transition probability is

$$P(s' \mid s, a) = \begin{cases} 1 & \text{for the unique next state } s', \\ 0 & \text{otherwise.} \end{cases}$$

3.4 Reward Function

We define a step cost of -1 for every non-terminal transition and a terminal reward of $+10$ for reaching the goal.[1] Let $R(s)$ denote the reward associated with state s:

$$R(s) = \begin{cases} 10 & \text{if } s = s_G, \\ -1 & \text{otherwise.} \end{cases}$$

3.5 Discount Factor

We use a discount factor

$$\gamma = 0.9,$$

which trades off immediate rewards against future rewards.[1]

4 Value Iteration Algorithm

Value iteration computes the optimal state-value function $V^*(s)$ by repeatedly applying the Bellman optimality operator.[1] The optimal value function is defined as

$$V^*(s) = \max_{\pi} V^{\pi}(s),$$

where V^{π} is the value function under policy π.[1]

4.1 Bellman Optimality Equation

For finite MDPs, V^* satisfies the Bellman optimality equation:

$$V^*(s) = \max_{a \in \mathcal{A}} \sum_{s'} P(s' \mid s, a) \left[R(s) + \gamma V^*(s') \right].$$

In our deterministic grid world, this simplifies to

$$V^*(s) = \max_{a \in \mathcal{A}} \left[R(s) + \gamma V^*(s') \right],$$

where s' is the unique next state resulting from taking action a in state s.[1]

4.2 Iterative Update Rule

Value iteration initializes $V_0(s)$ arbitrarily (often to zero) and updates values as

$$V_{k+1}(s) = \max_{a \in \mathcal{A}} \left[R(s) + \gamma V_k(s') \right].$$

For terminal states, we keep the value fixed or exclude them from updates.[1] The algorithm stops when the maximum change between iterations is below a small threshold θ:

$$\max_s |V_{k+1}(s) - V_k(s)| < \theta.$$

4.3 Extracting the Optimal Policy

After convergence, we can derive an optimal policy π^* from V^* by acting greedily with respect to the value function.[1] For each non-terminal state s,

$$\pi^*(s) = \arg\max_{a \in \mathcal{A}} \left[R(s) + \gamma V^*(s') \right].$$

In the grid world, this corresponds to choosing the action that points most directly along the optimal path to the goal, taking into account boundaries and the discount factor.

5 Python Implementation

This section provides a complete Python implementation of value iteration for the described 5x5 grid world.[4, 5]

5.1 Environment Setup

We first define the grid size, actions, movement deltas, rewards, and algorithm hyperparameters.

```
import numpy as np

# Grid size
rows = 5
cols = 5

# Actions: 0=up, 1=down, 2=left, 3=right
actions = ['^', 'v', '<', '>']

# Movement deltas for each action
deltas = [(-1, 0),  # up
          ( 1, 0),  # down
          ( 0,-1),  # left
          ( 0, 1)]  # right

# Rewards: -1 everywhere, +10 at goal
R = np.full((rows, cols), -1.0)
```

```
goal = (4, 4)
R[goal] = 10.0

# Initialize value function and policy
V = np.zeros((rows, cols))
policy = np.array([[' ' for _ in range(cols)] for _ in range(rows
    )])

# Discount factor and convergence threshold
gamma = 0.9
theta = 1e-6
```

5.2 Value Iteration Loop

We now implement the main value iteration loop. For each state, we evaluate all possible actions, compute their action-values, and keep the maximum.[4]

```
while True:
    delta = 0.0
    # Loop over all states
    for i in range(rows):
        for j in range(cols):

            # Skip updating the terminal goal state
            if (i, j) == goal:
                continue

            v_old = V[i, j]
            max_q = float('-inf')
            best_a = 0

            # Evaluate each action
            for a in range(4):
                di, dj = deltas[a]
                ni = i + di
                nj = j + dj

                # Check if next state is inside the grid
                if 0 <= ni < rows and 0 <= nj < cols:
                    next_v = V[ni, nj]
                else:
                    # If we hit a wall, we stay in the same cell
```

```
                    ni, nj = i, j
                    next_v = V[i, j]

                # Deterministic transition: Q-value for (s,a)
                q = R[i, j] + gamma * next_v

                # Keep the best action-value
                if q > max_q:
                    max_q = q
                    best_a = a

            # Update value and policy
            V[i, j] = max_q
            policy[i, j] = actions[best_a]

            # Track maximum change for convergence
            delta = max(delta, abs(v_old - V[i, j]))

    # Stop if the change is below the threshold
    if delta < theta:
        break
```

5.3 Printing the Optimal Policy

After convergence, we print the optimal policy for each state in the grid.[4]

```
print("Optimal Policy (5x5 grid, G = goal):")
for i in range(rows):
    row_symbols = []
    for j in range(cols):
        if (i, j) == goal:
            row_symbols.append('G')
        else:
            row_symbols.append(policy[i, j])
    print(' '.join(row_symbols))
```

A typical output will show arrows pointing towards the goal state from most grid cells, with "G" marking the terminal state.

6 Discussion

The value iteration algorithm converges to the optimal value function for the 5x5 grid world with the given reward structure and discount factor.[1] The resulting policy directs the agent along shortest paths to the goal, while accounting for the negative step cost and discounting of future rewards.[1]

Because the environment is deterministic and small, value iteration is computationally inexpensive and converges quickly.[2] For larger or stochastic environments, similar ideas apply, but we may need more efficient or approximate methods such as policy iteration or function approximation.[3]

7 Conclusion

This report demonstrated how to model a 5x5 grid world as an MDP and solve it using value iteration. We described the state and action spaces, transition dynamics, and reward function, derived the Bellman optimality update, and implemented the algorithm in Python to compute the optimal policy for each state.[4, 5]

References

[1] R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, 2nd edition, 2018.

[2] M. L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. Wiley, 1994.

[3] IntroRL Documentation, Chapter 3: Finite MDP. `https://introrl.readthedocs.io/en/latest/chapter_3.html`.

[4] GeeksforGeeks. Implement Value Iteration in Python. `https://www.geeksforgeeks.org/python/implement-value-iteration-in-python/`, 2024.

[5] Towards Data Science. Implement Value Iteration in Python - A Minimal Working Example. `https://towardsdatascience.com/implement-value-iteration-in-python-a-minimal-working-example-f638907f3437/`, 2025.