Reinforcement Learning: `GridWorld` Exploration

Gavin Hull
Memorial University of Newfoundland and Labrador
ghull@mun.ca

July 19, 2024

Introduction

Reinforcement learning (RL) is a fundamental paradigm in the field of artificial intelligence that enables agents to learn optimal behaviours through interaction with their environment. This project explores the application of various RL techniques to the classic `GridWorld` problem- a simple yet flexible testbed for evaluating RL algorithms.

We implement and analyze a range of methods for solving Markov Decision Processes (MDPs) in a 5×5 `GridWorld` environment including explicit solutions, iterative methods, and Monte Carlo approaches. By applying these techniques to both stationary and dynamic variants of the environment, we gain insight into their relative strengths, limitations, and practical applications.

This report presents our findings, comparing the performance and characteristics of different RL algorithms in solving the `GridWorld` problem. Through this analysis, we aim to deepen our understanding of core RL concepts, and contextualize their place in modern AI.

All code is made available through GitHub. [1]

Environment Setup

To facilitate our experiments with the `GridWorld` environment, we implement a custom `GridWorld` class. This provides the necessary flexibility and efficient representation required to adequately execute the desired algorithms.

Key design decisions include:

1. **Nested `Block` Class:** The `GridWorld` class contains a nested `Block` class which represents an individual cell within the grid. This encapsulation permits easy management of state transitions and reward distributions for each position within the grid.
2. **Array Imitation**: The grid is represented as a 2D `numpy` array of `Block` objects. In combination with over-written `__getitem__` and `__setitem__` methods, this enables efficient and convenient indexing and manipulation of the environment.
3. **Visualization Methods:** Built-in plotting functions for both state- and action-value functions strike a balance between ease of analysis and effective interpretation of results.

[1] All code required to replicate results can be found here: https://github.com/CallMeTwitch/Math-4250/tree/main/GridWorld

Part 1: Infinite Horizon `GridWorld` Environments

In this section we consider a simple 5×5 `GridWorld` environment with two special states as outlined in the assignment. Using three methods each, we will solve first for the value function, then for the optimal policy.

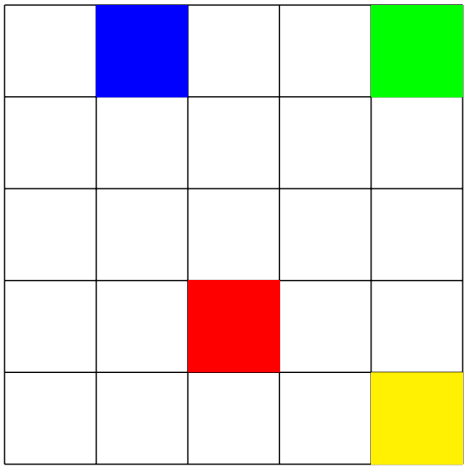

Figure 1: Environment grid A.

1.1 Value Function

We compute the value function with three distinct methods: explicit solution of the system of Bellman equations, iterative policy evaluation, and value iteration. Each method provides unique insight into the problem and serves as a foundation for later policy optimization.

1.1.1 Explicit Solution

The explicit solution method directly solves the system of Bellman equations using techniques from linear algebra. This approach provides an efficient route to the exact solution but becomes exponentially more computationally intensive as the state space increases. An explicit pseudo-code implementation can be found in Appendix A.

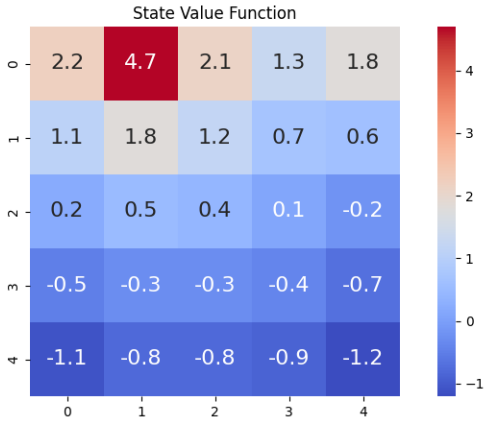

Figure 2: Heatmap of the value function with equiprobable policy in the 5×5 `GridWorld` environment, solved explicitly.

The solution shows peak values at the high-rewarding special states, with the highest value assigned to the cell with the highest reward. We also observe a gradient of decreasing values as distance from the special states increases, reflecting the impact of the discounted future reward. The only surprise we note in this figure is that the value of the two special states is lower than their immediate reward. We suspect that this signifies that with an equiprobable policy, an agent is likely to accumulate multiple fall-off penalties between occupations at the special states.

1.1.2 Iterative Policy Evaluation

Iterative policy evaluation computes the value function of a given policy by repeatedly applying the Bellman expectation equation until convergence. This method can be far more robust than the explicit solution when state spaces are increased but will often converge to approximate solutions. An explicit pseudo-code implementation can be found in Appendix B.

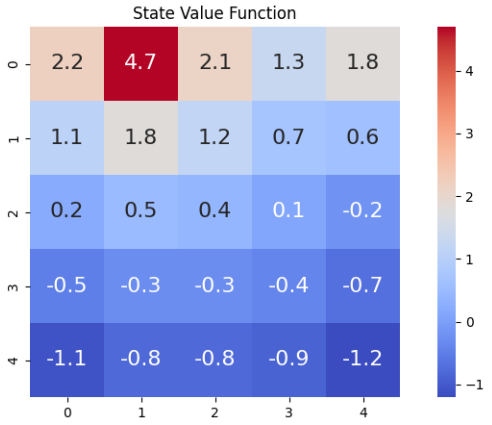

Figure 3: Heatmap of the value function with equiprobable policy in the 5×5 `GridWorld` environment, computed using iterative policy evaluation.

These results are a perfect copy of the explicit solution, demonstrating consistency of the value function across different computation methods and therefore validating the results of both methods.

1.1.3 Value Iteration

Value iteration is a dynamic programming algorithm that computes the *optimal* value function by iteratively improving the estimate of the best value for each state. An explicit pseudo-code implementation of the algorithm can be found in Appendix C.

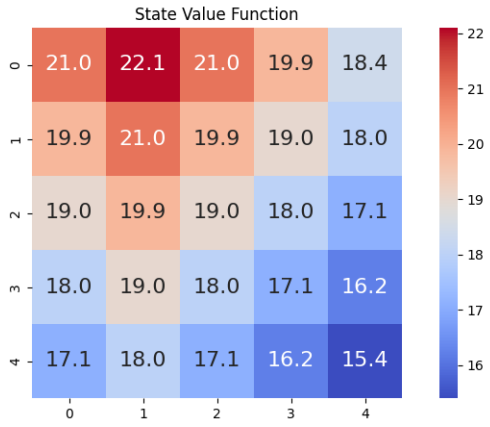

Figure 4: Heatmap of the optimal value function in the 5×5 `GridWorld` environment, computed using value iteration.

Surprisingly, value iteration reveals that the lesser special state (with reward 2.5) has a lower value than all others within 3 cells of the greater special state. We posit that this is likely due to a combination of its location in the corner (as it is adjacent to two potential fall-off penalties) and its potential transition to the lowest-value cell on the grid (the cell furthest from the greater special state.) Apart from this anomaly, the optimal value function exhibits a perfect gradient, decreasing near-linearly with distance from the greater special state.

1.2 Policy Function

We now turn our attention to determining the optimal policy for this `GridWorld` environment. We do so with three different methods: explicitly solving the Bellman Optimality Equation, policy iteration with iterative policy evaluation, and policy iteration with value iteration. Each method builds from those in the previous section, exhibiting many of the same features.

1.2.1 Explicit Solution

The explicit solution for the optimal policy is obtained by directly solving the Bellman Optimality Equation. Unlike the explicit solution of the value function, which can be efficiently solved with linear algebra, the explicit solution of the policy contains a max function. Therefore, we approach it as a root-finding problem, employing the L-BFGS-G algorithm for its quasi-Newtonian properties. An explicit pseudo-code implementation can be found in Appendix D.

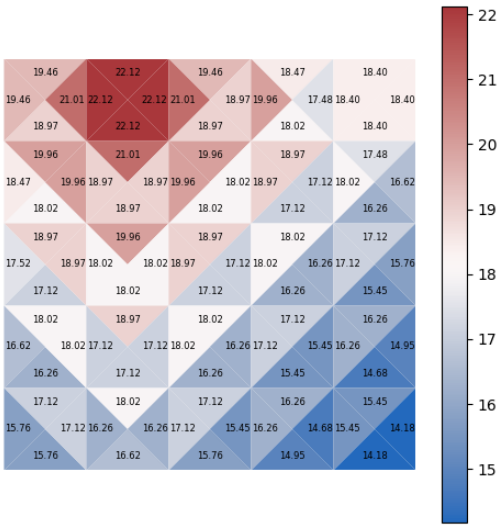

Figure 5: Optimal action-value function for the 5×5 `GridWorld` environment, solved explicitly.

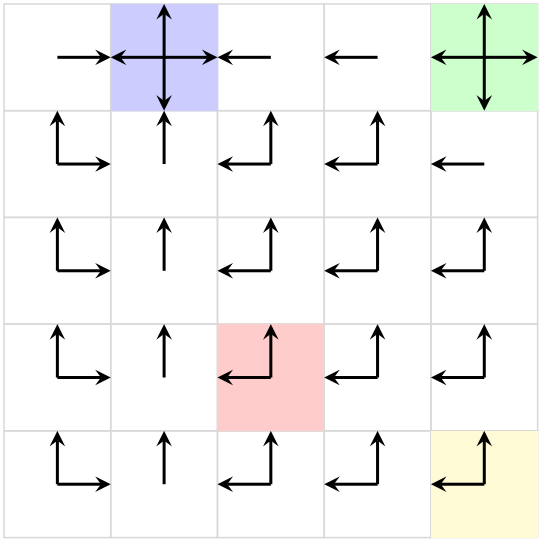

Figure 6: Optimal policy for the 5×5 `GridWorld` environment.

The optimal policy clearly shows preference for the greater special state, as evidenced by the gradient in the action-value function. Notably, the policy graph shows that the optimal policy actively avoids the lesser special state in favour of its higher rewarding cousin. This strategy is supported by the lack of penalty for travelling from white squares in this environment, meaning there is no penalty for travelling further in favour of a greater reward.

1.2.2 Policy Iteration

Policy iteration alternates between policy evaluation and policy improvement steps to converge on the optimal policy. An explicit pseudo-code implementation can be found in Appendix E.

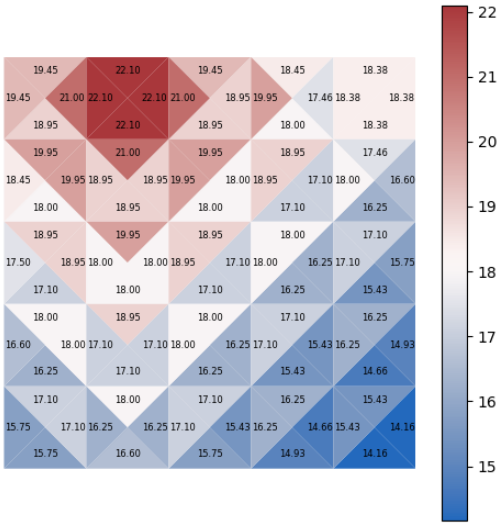

Figure 7: Optimal action-value function for the 5×5 `GridWorld` environment, computed using policy iteration.

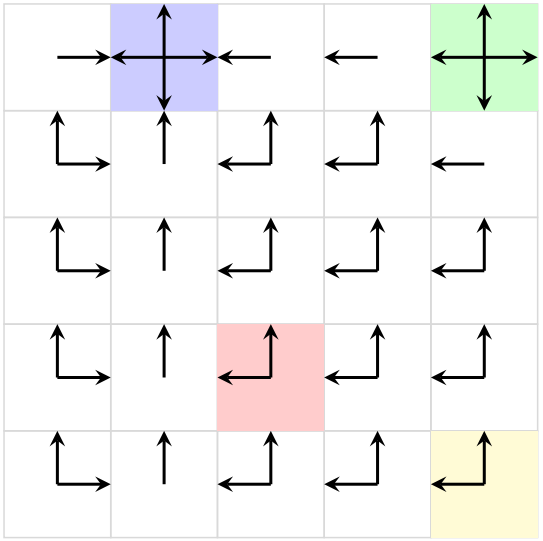

Figure 8: Optimal policy for the 5×5 `GridWorld` environment, computed using policy iteration.

The policy iteration method converges to an identical policy to the explicit solution, demonstrating further consistency between approaches and validating the results of the two methods.

1.2.3 Policy Improvement with Value Iteration

Policy Improvement with Value Iteration computes the optimal value function using value iteration and subsequently derives the optimal policy. An explicit pseudo-code implementation can be found in Appendix F.

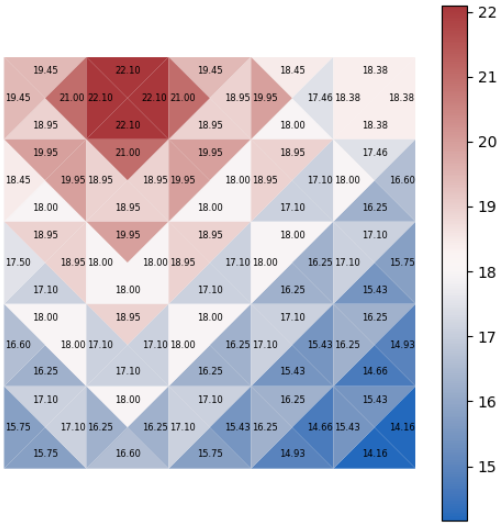

Figure 9: Optimal action-value function for the 5×5 `GridWorld` environment, derived from policy improvement with value iteration.

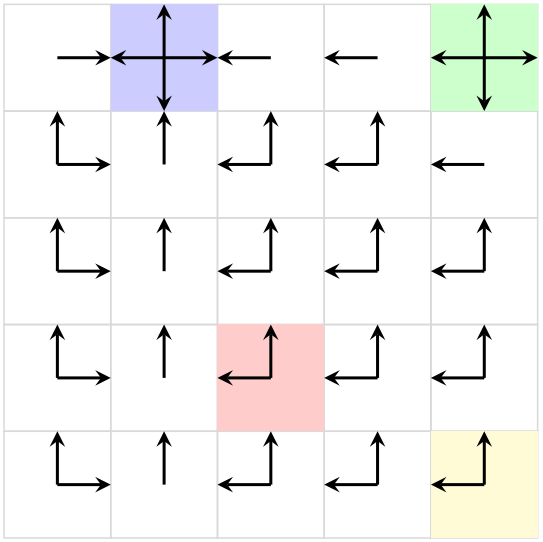

Figure 10: Optimal policy for the 5×5 `GridWorld` environment, derived using policy improvement with value iteration.

The policy derived via value iteration matches the policies obtained by the other methods, confirming the consistency of the optimal policy across different solution techniques.

All three methods result in the same optimal policy, which directs the agent toward the highest rewarding states while avoiding penalties. The policy shows a clear pattern of actions leading to the highest value state (with reward 5) from all other states except for the special state with reward 2.5, which has its own optimal action due to its unique transition probabilities. This behaviour is expected, as the optimal policy will maximize the expected reward over time.

Part 2: Terminating `GridWorld` Environments

In this section we alter the environment as displayed below and described in the assignment instructions.

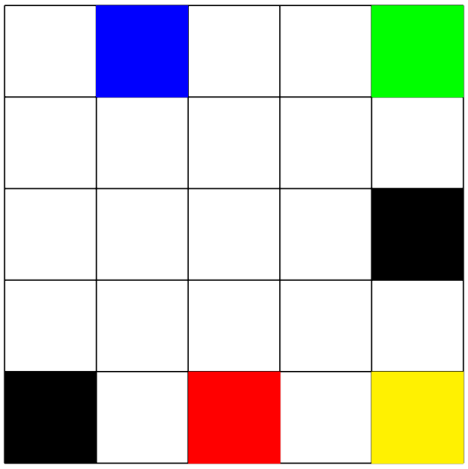

Figure 11: Environment grid B.

2.1 Monte Carlo Methods

We implement three Monte Carlo methods to learn optimal policies for the `GridWorld` environment: Monte Carlo with Exploring Starts, Monte Carlo with ϵ-Soft Policy, and Off-Policy Monte Carlo with Behaviour Policy.

2.1.1 Exploring Starts

The Monte Carlo method with exploring starts ensures that all state-action pairs have a non-zero probability of being selected by randomly choosing the initial state and action when initializing the episode. An explicit pseudo-code implementation of this algorithm can be found in Appendix G.

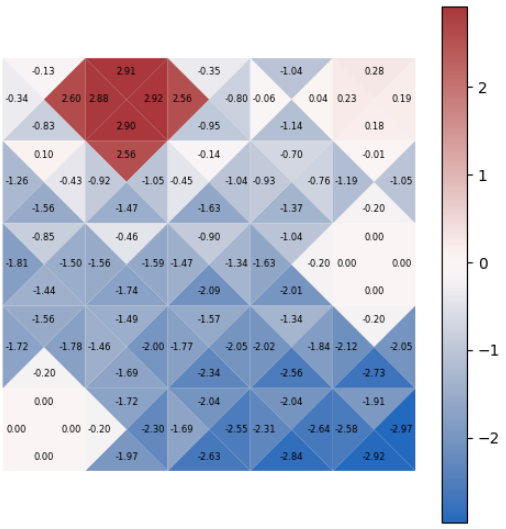

Figure 12: Optimal action-value function for the 5×5 `GridWorld` environment, approximated using Monte Carlo with Exploring Starts.

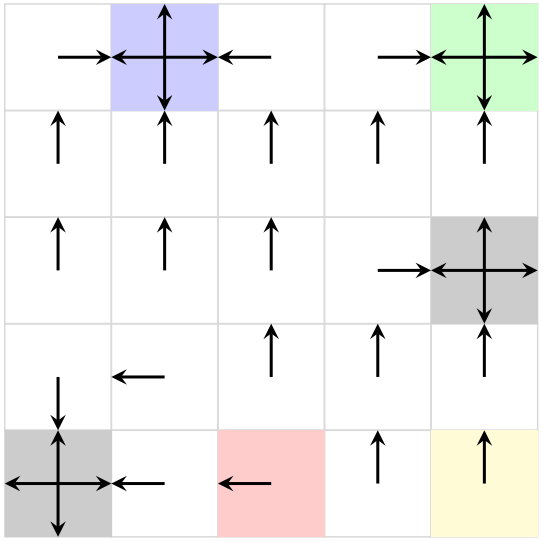

Figure 13: Optimal policy for the 5×5 `GridWorld` environment, approximated using Monte Carlo with Exploring Starts.

The optimal policy learned through Monte Carlo with Exploring Starts shows a clear pattern of actions leading toward the special states. However, some action-value pairs do exhibit unexpected behavior, such as moving left from cell (1, 1) when the high-rewarding blue cell is so close. Further analysis of the action-value function reveals that many values are compressed into a small range, suggesting we may be experiencing incomplete convergence. This may be due to insufficient episodes or suboptimal hyperparameters. These results will later be compared with those of other Monte Carlo methods to verify that there were in fact convergence issues.

2.1.2 ϵ-Soft Policy

The ϵ-soft policy ensures continuous exploration by maintaining a non-zero probability of selecting a random action (and subsequently a non-zero probability of each state-action pair) given any state. An explicit pseudo-code implementation can be found in Appendix H.

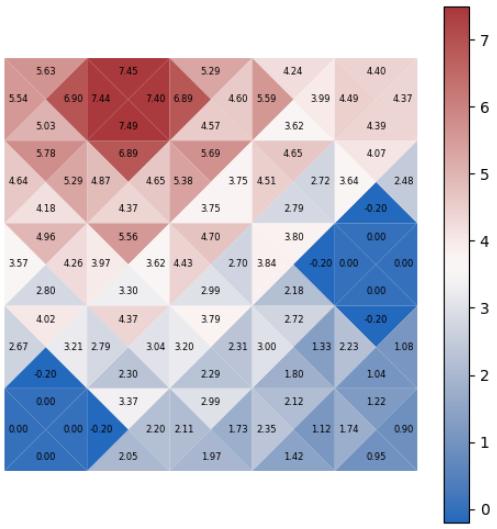

Figure 14: Optimal action-value function for the 5×5 `GridWorld` environment, approximated using Monte Carlo with ϵ-Soft Policy.

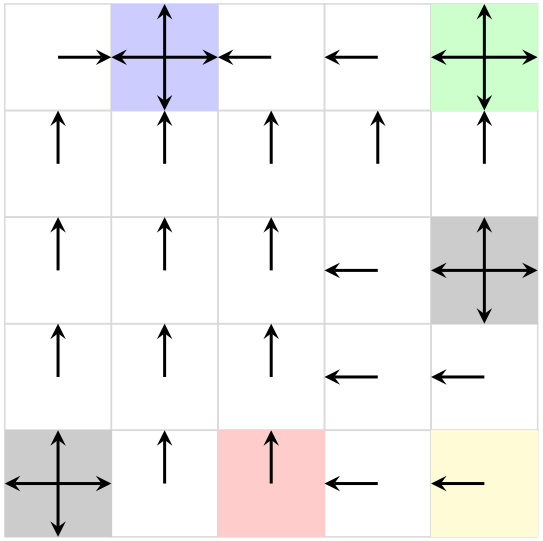

Figure 15: Optimal policy for the 5×5 `GridWorld` environment, approximated using Monte Carlo with ϵ-Soft Policy.

As anticipated, the ϵ-soft policy Monte Carlo method results in a different "optimal" policy than the Monte Carlo method with exploring starts, which is consistent with our theory that the Monte Carlo with Exploring Starts algorithm did not properly converge.

Contrary to the previous method, Monte Carlo with ϵ-Soft Policy results in a clean gradient from the greater special state. We see that the terminal states have a value of 0 with any action, since no matter which action is chosen the episode will then end, and actions resulting in moving to terminal states have the lowest value of any action-state pair. The optimal policy travels to the greater special states from every cell in the grid aside from (4, 3) which travels to the lesser special state. Given that the lesser special state rewards half that of the greater special state and has a 50% chance of sending the agent to the furthest cell from the greater special state, we hypothesize that this is also an artifact of non-convergence. With some minor hyperparameter tuning — for example, increasing the number of episodes and potentially decreasing ϵ — we posit that it would converge to direct toward the greater special state along with the rest of the cells.

2.1.3 Behaviour Policy

Off-policy learning allows the agent to learn from experiences generated by a different policy than the one being optimized. This may be convenient in real-world scenarios where a "good" policy is difficult to find, but it also tends to be unstable. Explicit pseudo-code can be found in Appendix I.

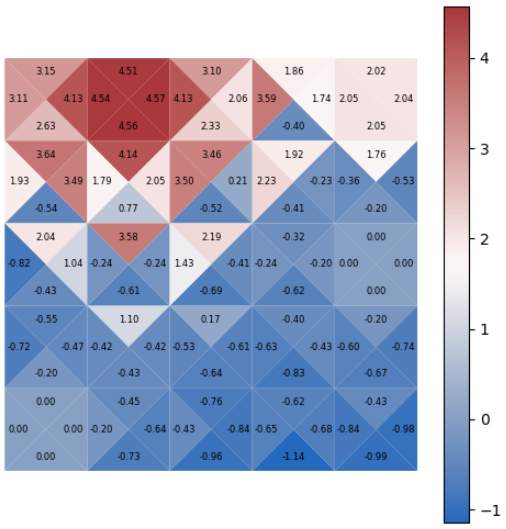

Figure 16: Optimal action-value function for the 5×5 `GridWorld` environment, computed with behaviour policy.

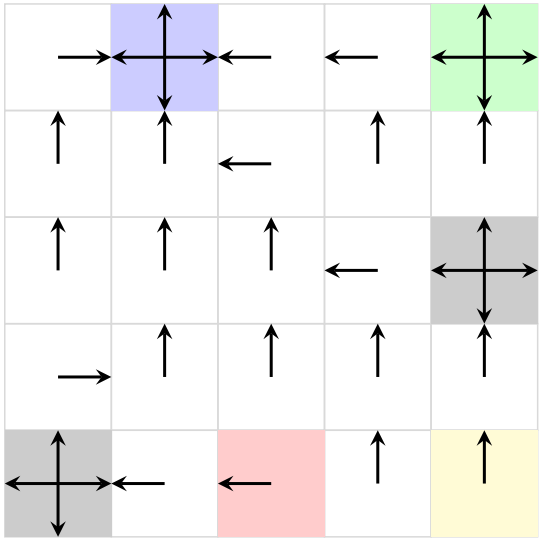

Figure 17: Optimal policy for the 5×5 `GridWorld` environment.

To our great surprise, we find another distinct "optimal" action-value function with the off-policy Monte Carlo method. This further confirms our suspicion that the Monte Carlo methods have difficulty converging- likely due to their inherent reliance on randomness.

The derived policy does tend toward the greater special state as expected; however, the action-value grid shows little variance in the values on the lower half of the grid. Once again, we suspect that given more episodes to converge (and perhaps given a more optimal behaviour policy) this method would also converge to a truly optimal action-value function and thus optimal policy.

2.2 Dynamic Environment

In this section, we introduce stochasticity to the environment, randomly swapping the two special states with probability 0.1 at each time step as outlined in the assignment. We solve for the optimal policy using two variations on policy iteration.

2.2.1 Monte Carlo Transition Matrix Approximation

We approximate the transition matrix using Monte Carlo sampling and then apply standard policy iteration. Despite potential instability, Monte Carlo methods are necessary here as the agent lacks information about special state locations at each time step. This approach allows us to estimate the transition matrix from actual samples over time.

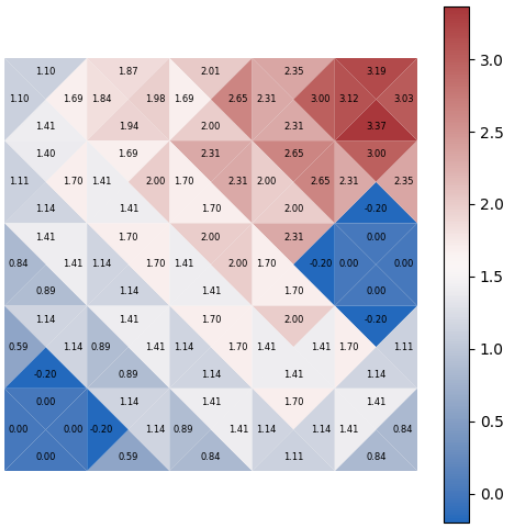

Figure 18: Optimal action-value function for the 5×5 `GridWorld` environment, computed using policy iteration with Monte Carlo sampling.

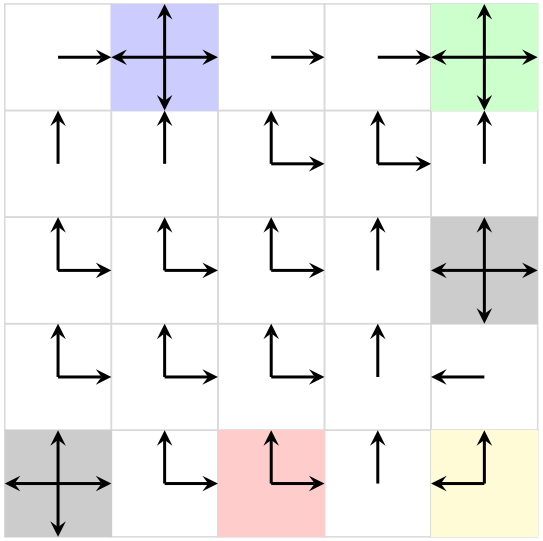

Figure 19: Optimal policy for the 5×5 `GridWorld` environment.

Surprisingly, the optimal policy appears to consistently favor the lesser special state. This unexpected result likely stems from the inherent randomness in Monte Carlo methods used to approximate the transition matrix. Multiple iterations of this process may yield different convergence points, sometimes favoring the greater special state. This highlights the sensitivity of Monte Carlo methods to initial conditions and sampling variance, especially in dynamic environments.

2.2.2 Time-Aware Algorithm

In this section we introduce time step information to the state of the environment. Given this knowledge, we can explicitly calculate the probability of the two special states being swapped, then use standard policy iteration to find the optimal policy at a given time step. This circumvents the need for Monte Carlo methods and maintains stability in the iteration process. The derivation of the formulas describing the probability of the two states being swapped at any given time can be found in Appendix J.

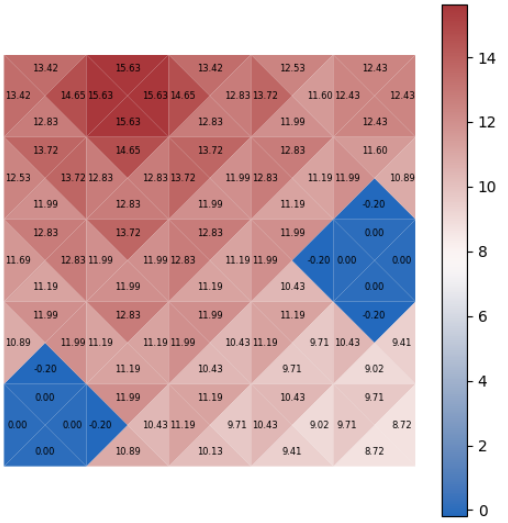

Figure 20: Optimal action-value function at time $t = 0$ for the 5×5 `GridWorld` environment, computed with time-aware policy iteration.

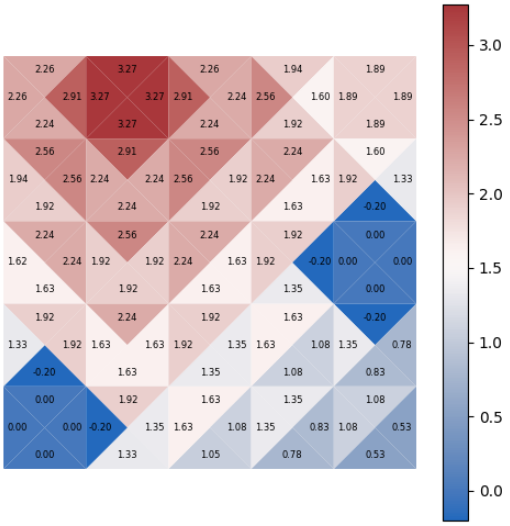

Figure 21: Optimal action-value function at $t \geq 100$ for the 5×5 `GridWorld` environment, computed with time-aware policy iteration.

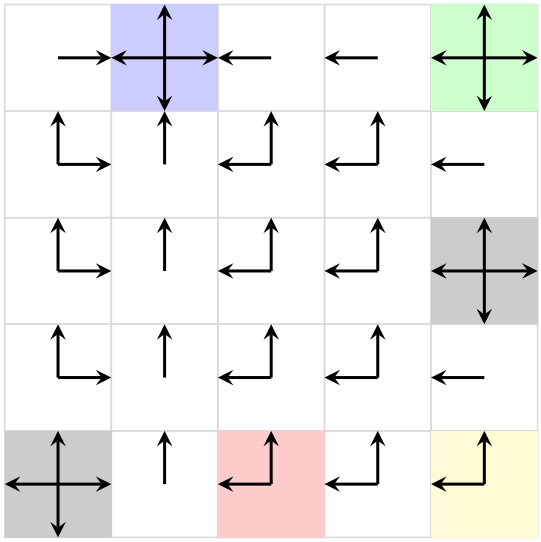

Figure 22: Optimal policy for the 5×5 `GridWorld` environment, over all time steps (no change was exhibited).

In Figure 20 we see the action-value function for $t = 0$, while Figure 21 displays the action-value function for $t = 100$. We note that from $t = 100$ to $t = 1,000$ there was no change in the action value function. As anticipated, the greater special state is favoured in the optimal policy for all time steps. In fact, the only difference we see between time steps is a decrease in the severity of the gradient, which is likely due to the uncertainty of the position of the greater special state over time. Over all time steps we see that the optimal policy travels toward the cell instantiated with the greater special state. This is consistent with our derivation of the probability of the two states having swapped, as $\frac{1-0.8^t}{2} \leq \frac{1+0.8^t}{2}, \forall t \geq 0$ (see Appendix J.)

3 Conclusion

Our exploration of the `GridWorld` environment using various RL techniques has revealed several key insights. Among infinite horizon environments, explicit solutions and iterative methods produce consistent results, validating their effectiveness in stationary environments. Meanwhile, in terminating environments Monte Carlo approaches demonstrated problems converging, highlighting limitations due to their inherent randomness.

Generally, optimal policies favoured paths toward the greater special state, tending to avoid the lesser special state as well as terminal states. As expected, these policies often minimized the number of moves required of an agent to arrive at the greater special state. Finally, our time-aware algorithm showed great promise, maintaining the accuracy and efficiency of the policy iteration method despite the introduced stochasticity.

Each method presented unique trade-offs between accuracy, simplicity, and computational efficiency. Future work may explore the addition of ML techniques to pre-existing algorithms or further generalize our time-aware policy iteration method to a matrix of continuous functions rather than a discrete collection of matrices.

This exploration of `GridWorld` through various RL methods demonstrates both the power and challenges of reinforcement learning, highlighting its potential applications in complex real-world scenarios and its potential in the development of artificial general intelligence going forward.

References

[1] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

Appendices

All algorithms described in these appendices are from Sutton. [1]

Appendix A

Algorithm 1 Explicit Solution for the System of Bellman Equations

1: Compute state transition probabilities:

$$p(s'|s) = \sum_{a,r} \pi(a|s)p(s', r|s, a)$$

2: Compute expected rewards:

$$R(s) = \sum_{a,r,s'} \pi(a|s)p(s', r|s, a)R(s, a)$$

3: Solve for value function:

$$V = (I - \gamma p)^{-1} R$$

Appendix B

Algorithm 2 Iterative Policy Evaluation

1: Initialize $V(s)$ arbitrarily for all states s
2: **repeat**
3: $\quad \Delta \leftarrow 0$
4: $\quad$ **for** each state s **do**
5: $\quad\quad v \leftarrow V(s)$
6: $\quad\quad V(s) \leftarrow \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)[r + \gamma V(s')]$
7: $\quad\quad \Delta \leftarrow \max(\Delta, |v - V(s)|)$
8: $\quad$ **end for**
9: **until** $\Delta < \theta$ (a small threshold)

Appendix C

Algorithm 3 Value Iteration

1: Initialize $V(s)$ arbitrarily for all states s
2: **repeat**
3: $\quad \Delta \leftarrow 0$
4: $\quad$ **for** each state s **do**
5: $\quad\quad v \leftarrow V(s)$
6: $\quad\quad V(s) \leftarrow \max_a \sum_{s',r} p(s', r|s, a)[r + \gamma V(s')]$
7: $\quad\quad \Delta \leftarrow \max(\Delta, |v - V(s)|)$
8: $\quad$ **end for**
9: **until** $\Delta < \theta$ (a small threshold)

Appendix D

Algorithm 4 Explicit Solution for Bellman Optimality Equation

Define loss function for value estimates: $\text{loss}(V) = \text{MSE}(V, \max_a \sum_{s',r} p(s', r|s, a)[r + \gamma V(s')])$
Minimize loss function using L-BFGS-B method
Compute Q-values: $Q(s, a) = \sum_{s',r} p(s', r|s, a)[r + \gamma V(s')]$
Determine optimal policy: $\pi^*(s) = \text{argmax}_a Q(s, a)$

Appendix E

Algorithm 5 Policy Iteration (using iterative policy evaluation)

Initialization
$V(s) \in \mathbb{R}$ and $\pi(s) \in \mathcal{A}(s)$ arbitrarily for all $s \in S; V(\text{terminal}) \doteq 0$
Policy Evaluation
repeat
    $\Delta \leftarrow 0$
    for each $s \in S$ **do**
        $v \leftarrow V(s)$
        $V(s) \leftarrow \sum_{s',r} p(s', r|s, \pi(s))[r + \gamma V(s')]$
        $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
    end for
until $\Delta < \theta$ (a small positive number determining the accuracy of estimation)
Policy Improvement
policy_stable $\leftarrow$ true
for each $s \in S$ **do**
    old_action $\leftarrow \pi(s)$
    $\pi(s) \leftarrow \arg\max_a \sum_{s',r} p(s', r|s, a)[r + \gamma V(s')]$
    if *old_action* $\neq \pi(s)$ **then**
        policy_stable $\leftarrow$ false
    end if
end for
if *policy_stable* **then**
    stop and return $V \approx v_*$ **and** $\pi \approx \pi_*$
else
    go to 3
end if

Appendix F

Algorithm 6 Policy Improvement with Value Iteration

Initialize $V(s)$ arbitrarily for all states s
repeat
    $\Delta \leftarrow 0$
    for each state s **do**
        $v \leftarrow V(s)$
        $V(s) \leftarrow \max_a \sum_{s',r} p(s', r|s, a)[r + \gamma V(s')]$
        $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
    end for
until $\Delta < \theta$ (a small threshold)
$\pi^*(s) \leftarrow \text{argmax}_a \sum_{s',r} p(s', r|s, a)[r + \gamma V(s')]$

Appendix G

Algorithm 7 Monte Carlo with Exploring Starts

Initialize:
  $\pi \leftarrow$ an arbitrary policy
  $Q(s,a) \in \mathbb{R}$ (arbitrarily), for all $s \in \mathcal{S}, a \in \mathcal{A}(s)$
  Returns$(s,a) \leftarrow$ empty list, for all $s \in \mathcal{S}, a \in \mathcal{A}(s)$
repeat(for each episode):
    Generate an episode following π with random S_0, A_0: $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$
    $G \leftarrow 0$
  for each step of episode, $t = T-1, T-2, \ldots, 0$: **do**
      $G \leftarrow \gamma G + R_{t+1}$
    if the pair S_t, A_t does not appear in $S_0, A_0, S_1, A_1, \ldots, S_{t-1}, A_{t-1}$ **then**
        Append G to Returns(S_t, A_t)
        $Q(S_t, A_t) \leftarrow$ average(Returns(S_t, A_t))
    end if
  end for
until convergence
$\pi \leftarrow \text{argmax}_a Q(s,a)$

Appendix H

Algorithm 8 Monte Carlo with ϵ-Soft Policy

Algorithm parameter: small $\varepsilon > 0$
Initialize:
  $\pi \leftarrow$ an arbitrary ε-soft policy
  $Q(s,a) \in \mathbb{R}$ (arbitrarily), for all $s \in \mathcal{S}, a \in \mathcal{A}(s)$
  Returns$(s,a) \leftarrow$ empty list, for all $s \in \mathcal{S}, a \in \mathcal{A}(s)$
repeat(for each episode):
    Generate an episode following π: $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$
    $G \leftarrow 0$
  for each step of episode, $t = T-1, T-2, \ldots, 0$: **do**
      $G \leftarrow \gamma G + R_{t+1}$
    if the pair S_t, A_t appears in $S_0, A_0, S_1, A_1, \ldots, S_{t-1}, A_{t-1}$ **then**
        Append G to Returns(S_t, A_t)
        $Q(S_t, A_t) \leftarrow$ average(Returns(S_t, A_t))
    end if
  end for
until convergence

Appendix I

Algorithm 9 Off-policy Monte Carlo with Behaviour Policy

Initialize, for all $s \in \mathcal{S}, a \in \mathcal{A}(s)$:
    $Q(s,a) \in \mathbb{R}$ (arbitrarily)
    $C(s,a) \leftarrow 0$
    $\pi(s) \leftarrow \arg\max_a Q(s,a)$ (with ties broken consistently)
loop
    $b \leftarrow$ any soft policy
    Generate an episode using $b : S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$
    $G \leftarrow 0$
    $W \leftarrow 1$
    for each step of episode, $t = T-1, T-2, \ldots, 0$ **do**
        $G \leftarrow \gamma G + R_{t+1}$
        $C(S_t, A_t) \leftarrow C(S_t, A_t) + W$
        $Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \frac{W}{C(S_t, A_t)}[G - Q(S_t, A_t)]$
        $\pi(S_t) \leftarrow \arg\max_a Q(S_t, a)$ (with ties broken consistently)
        if $A_t \neq \pi(S_t)$ **then**
            break
        end if
        $W \leftarrow W \cdot \frac{1}{b(A_t|S_t)}$
    end for
end loop

Appendix J

To derive the explicit formula for the probability of the two special states being swapped at any given time step t, we use combinatorics.

At time step t, there will have been t opportunities for the two special states to have swapped cells with probability 0.1. If they have swapped an even number of times, then they will have returned to their original positions. Conversely, if they have swapped an odd number of times then they will be in each others cells. Then by basic sum and product rules, we find

$$P_O(t) = \sum_{\text{even } n \geq 0}^{t} \binom{t}{n} (0.9)^{t-n} (0.1)^n \tag{1}$$

and

$$P_S(t) = \sum_{\text{odd } n \geq 0}^{t} \binom{t}{n} (0.9)^{t-n} (0.1)^n \tag{2}$$

where $P_O(t)$ is the probability at time t that the two states are in their Original positions, and $P_S(t)$ is the probability at time t that the two states have Swapped.

Then note that by the binomial theorem we have

$$1 = (0.9 + 0.1)^t = \sum_{n=0}^{t} \binom{t}{n} (0.9)^{t-n} (0.1)^n \tag{3}$$

and

$$0.8^t = (0.9 - 0.1)^t = \sum_{n=0}^{t} \binom{t}{n} (-1)^n (0.9)^{t-n} (0.1)^n \tag{4}$$

Then adding and subtracting these two equations, we find

$$P_O(t) = \frac{1 + 0.8^t}{2} \tag{5}$$

and

$$P_S(t) = \frac{1 - 0.8^t}{2} \tag{6}$$

Therefore, the probability of the two states being in their original positions at time t is $\frac{1+0.8^t}{2}$, whereas the probability of their positions being swapped is $1 - \frac{1+0.8^t}{2} = \frac{1-0.8^t}{2}$. We use these formulas to perform policy iteration across all time steps of the dynamic `GridWorld` environment in parallel.