

Solving Markov Decision Processes and Multi-Armed Bandit Problems

Mikey Ferguson

April 21, 2025

1 Introduction

This project explores algorithms for sequential decision-making under uncertainty, focusing on both Markov Decision Processes (MDPs) and Multi-Armed Bandit (MAB) problems. We implement and compare several algorithms, including Value Iteration (VI), Modified Policy Iteration (MPI), Q-learning, SARSA, and the Upper Confidence Bound (UCB1) and ϵ -greedy strategies for MABs. The goal is to evaluate their performance, convergence behavior, and ability to scale across different environments.

2 Solving MDPs with a Known World Model

2.1 Problem Setup

We begin by solving MDPs where the transition and reward models are fully known. The two environments tested include:

- A 4x3 grid world with terminal and pit states.
- A modified Wumpus World with probabilistic pit deaths, variable step costs, and goal-return rewards.

2.2 Algorithms

We implement both Value Iteration and Modified Policy Iteration as described in Chapter 17. The Bellman update equations are applied iteratively until convergence in the case of value iteration or policy stability in the case of modified policy iteration.

$$V(s) = \max_a \left(\sum_{s'} P(s'|s, a) (r(s, a, s') + \gamma \cdot V(s')) \right)$$

After convergence, the policy garnered from value iteration is defined by:

$$\pi(s) = \operatorname{argmax}_a \left(\sum_{s'} P(s'|s, a) (r(s, a, s') + \gamma \cdot V(s')) \right)$$

In contrast, modified policy iteration for a given number of iterations, the value of each state is determined by the current policy:

$$V(s) = \sum_{s'} (P(s'|s, \pi(s)) \cdot (R(s, \pi(s), s') + \gamma \cdot V(s')))$$

And after a certain number of iterations, the policy is updated to:

$$\pi(s) = \sum_{s'} (P(s'|s, \pi(s)) \bullet (R(s, \pi(s), s') + \gamma \bullet V(s')))$$

And the previous two steps are repeated until the policy does not change.

2.3 Results and Analysis

For the classic grid world from the textbook, I observed the following results:

Value Iteration	Modified Policy Iteration

Noticeably, value iteration failed to reach the goal, while modified policy iteration not only succeeded but did so in fewer moves.

3 Learning with Unknown World Model

3.1 Problem Setup

The next phase of the project involved using reinforcement learning in three environments:

- 4x3 grid world
- 10x10 world with goal at (10,10)
- 10x10 world with goal at (5,5) (witnessed *utter* failure)

3.2 Algorithms

Q-learning and SARSA are implemented using both tabular methods and linear function approximation.

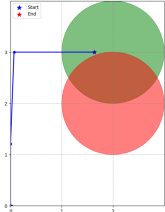
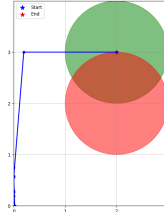
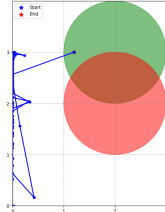
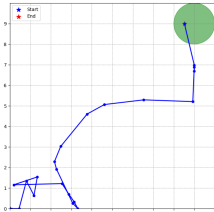
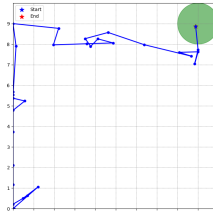
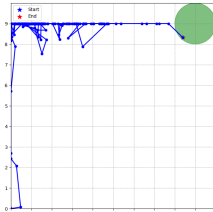
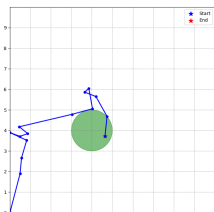
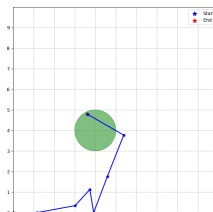
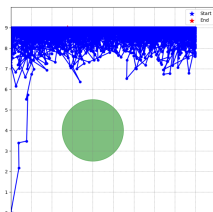
For table-based Q-learning in these continuous grid-world environments, it was necessary to discretize each state before referencing the table. My initial efforts ended absolutely horribly, and my lack of ideas for which features to use for the linear approximation as well as how to more effectively discretize the continuous environment did not help.

Luckily, Chat-GPT gave me some ideas. Unluckily, they still failed for the third environment. The main things to note are:

- States were discretized by seeing which “block” they were a part of when dividing the environment into fifty discrete blocks
- A state’s feature vector for the linear function approximation of a state/action pair used:
 1. Distance to goal
 2. Inverse distance to goal
 3. Boolean for whether the action will move the agent towards the goal (using the dot product of the actions direction with the direction to the goal)
 4. X-position divided by environment width
 5. Y-position divided by environment height
 6. X-component of the action direction
 7. Y-component of the action direction

3.3 Results and Analysis

Results were mixed, but much better for linear approximation methods. I (per ChatGPT’s suggestion) also included a greedy algorithm that simply moved closer to the goal as long as the agent was not moving into an obstacle. Note that because the greedy policy always reached the goal, it is *possible* to reach the goal.

	Greedy	Q-Table	Approximate Q-Learning
First Environment			
Second Environment			
Third Environment			

4 Multi-Armed Bandit Problems

4.1 Problem Setup

We simulate MAB environments and evaluate algorithms using:

- UCB
- ε -greedy with $\varepsilon = 0.1, 0.2, 0.3$

The heuristic for UCB is as follows:

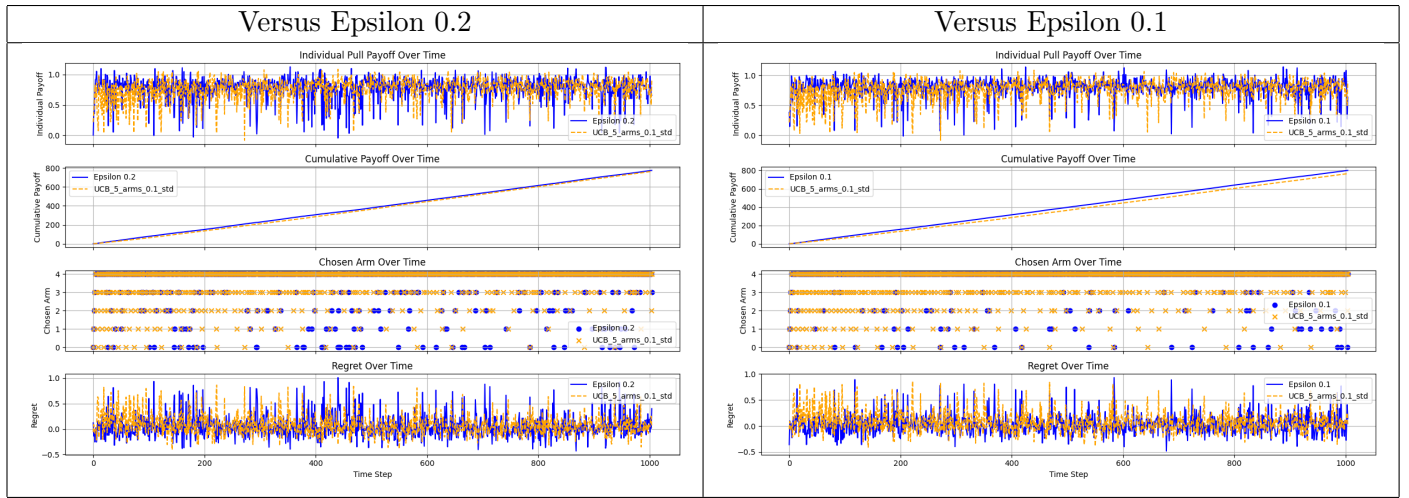
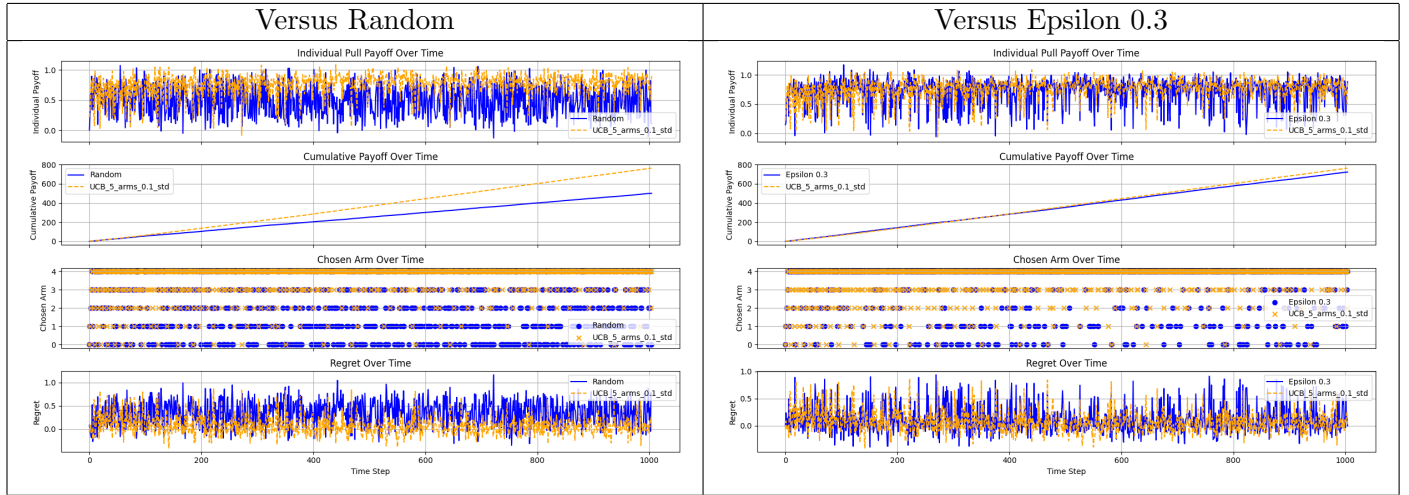
$$H(a) = \overline{\text{reward}(a)} + \sqrt{\frac{2\ln(n)}{n_a}}$$

4.2 Evaluation Metrics

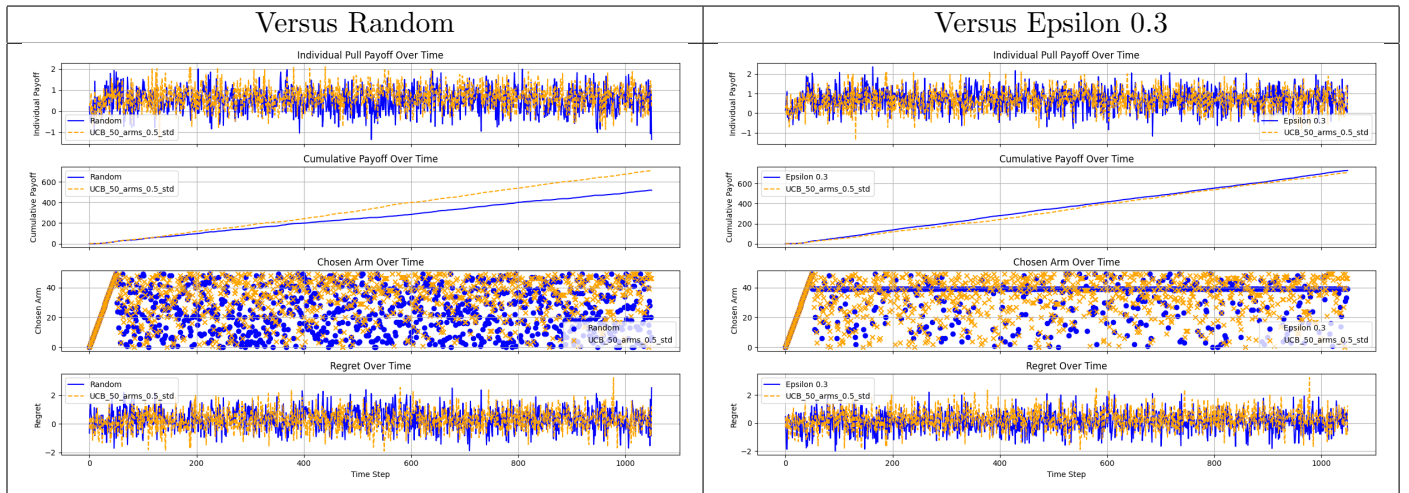
Regret, cumulative payoff, and estimated best arm over time.

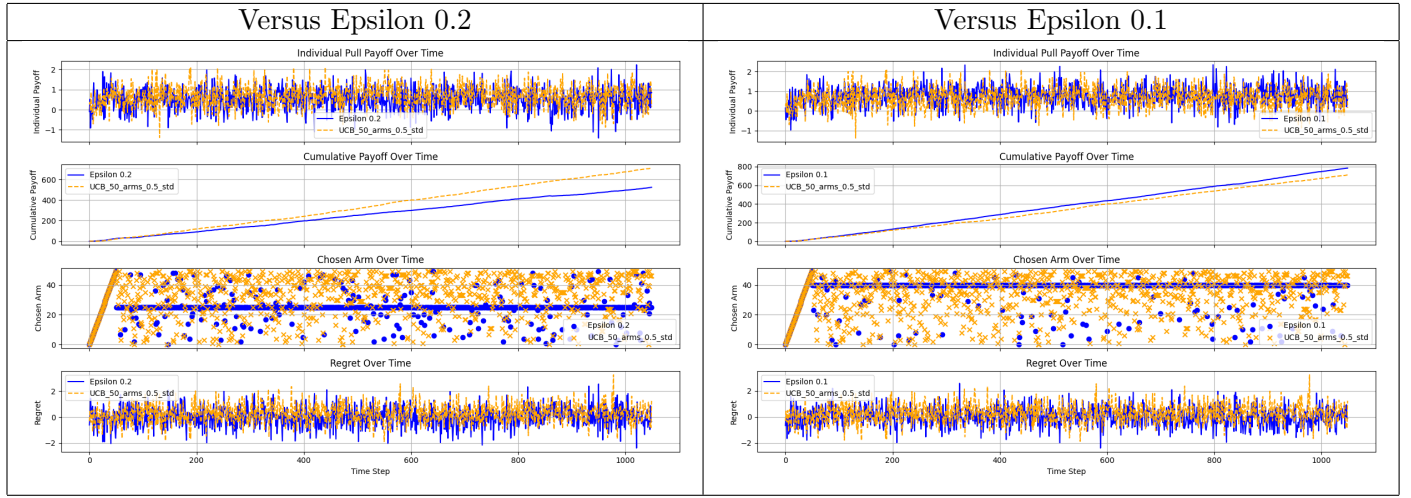
4.3 Results

When using a bandit simulator with 5 arms with a 0.1 standard deviation for the payout of each arm, I observed the following results:



Later, when using a bandit simulator with 50 arms with a 0.5 standard deviation for the payout of each arm, I observed similar results:





Ultimately, having randomness is dominated by all other strategies, and lower ϵ values for epsilon-greedy approaches (farther away from randomness) achieved better results.

Not surprisingly, UCB dominated random selection in both scenarios. It performed similarly to all epsilon-greedy selections, except – strikingly – the epsilon value of 0.2 in the case of 50 arms with a standard deviation of 0.5. This was noticeably outperformed by UCB, even though all other epsilon approaches either matched or slightly outperformed UCB.

Regret was consistent with performance - algorithms which achieved a better cumulative reward over time also achieved lower regret, as is to be expected.