SI140A Probability and Statistics Final Project: Performance Evaluation of Bandit Learning Algorithms

Anrui Wang Zhao Lu Jingran Fan

Due date: 2025/1/12 10:59pm

Abstract

Multi-armed bandit problems are fundamental models in sequential decision-making under uncertainty, wherein an agent must choose from several options (arms) over repeated trials to maximize cumulative rewards. These problems capture the delicate balance between exploration—seeking information about less-known options—and exploitation—leveraging current knowledge to select the best option. Classical bandit learning algorithms, such as ϵ-greedy, Upper Confidence Bound (UCB), and Thompson Sampling (TS), have been extensively studied and form the cornerstone of modern reinforcement learning techniques. More recently, Bayesian approaches that incorporate prior beliefs and update them with observed data have gained traction, offering theoretical elegance and robust performance in a variety of settings.

This project focuses on evaluating the performance of well-known bandit algorithms through numerical experiments. In Part I, we consider classical bandit algorithms operating on Bernoulli arms with parameters provided by an oracle. Although the oracle's parameters and optimal attainable reward (the "oracle value") are unknown to the algorithms, they provide a ground truth reference for performance comparison. We implement and benchmark ϵ-greedy (with various ϵ-values), UCB (with different confidence scales), and TS (with varying Beta priors) under identical experimental conditions. By analyzing their regret—defined as the gap between the algorithm's cumulative reward and the oracle value—we investigate how tuning parameters and prior knowledge impacts the exploration-exploitation trade-off.

In the optional Part II, we extend our analysis to a Bayesian bandit setting with discounted rewards, where prior distributions on arm parameters are continuously updated as more data is observed. We examine intuitive heuristics and discuss why these heuristics may fail to achieve optimality. Furthermore, we explore the derivation of optimal policies through recursive equations and investigate practical methods for exact and approximate solutions.

Overall, this project aims to provide a rigorous empirical evaluation of both classical and Bayesian bandit algorithms. By comparing their performance and understanding their underlying trade-offs, we gain deeper insights into bandit learning theory and develop intuition for selecting and designing effective strategies in diverse applications.

Contents

Introduction

In many real-world scenarios—from online advertising to medical trials—decision-makers must choose actions to maximize cumulative rewards. However, these decisions also provide valuable information that can guide future actions. This creates a fundamental tension between exploiting what we already know to gain immediate benefits and exploring new options to potentially improve future outcomes. This challenge is central to the field of reinforcement learning and is known as the exploration-exploitation trade-off.

A classic illustration is the multi-armed bandit problem. Imagine walking into a casino and facing a slot machine with multiple arms, each offering a different, unknown payoff distribution. Your goal is to pull the arms over a sequence of trials to earn as many rewards as possible. Since you do not know which arm is best, you must try them out (exploration) while continuing to play the arm that seems most promising (exploitation). Crucially, the reward probabilities remain fixed but hidden, and the only way to learn them is by experimenting.

This report examines three classical strategies for solving the multi-armed bandit problem—ϵ-greedy, Upper Confidence Bound (UCB), and Thompson Sampling (TS). By comparing their performance, we gain insight into how they balance exploration and exploitation and how well they adapt to uncertain, reward-driven environments.

Part I: Classical Bandit Algorithms

Problem 1

Choose $N = 5000$ and compute the theoretically maximized expectation of aggregate rewards over N time slots. Suppose we have an oracle that provides the parameters of the Bernoulli distributions for three arms as follows:

$$\theta_1 = 0.7, \quad \theta_2 = 0.5, \quad \theta_3 = 0.4.$$

We choose the time horizon as $N = 5000$ time steps. If we know these parameters beforehand (as the oracle does), the strategy to maximize the expected total reward is to always pull the arm with the highest success probability, which in this case is arm 1 (with $\theta_1 = 0.7$).

The expected reward is:

$$\max_{I(t), t=1,\ldots,N} \mathbb{E}\left[\sum_{t=1}^{N} r_{I(t)}\right] = N \times \theta_1 = 5000 \times 0.7 = 3500.$$

Thus, the theoretically maximized expectation is: $\boxed{3500.}$

Problem 2

Imports and parameters:

```
import matplotlib.pyplot as plt
import numpy as np
import random, math, copy

np.random.seed(42)
num_arms = 3
theta = np.array([0.7, 0.5, 0.4])
```

Implementation of ϵ-greedy algorithm:

```
def epsilon_greedy(epsilon, N, theta):
    Q = np.zeros(num_arms)  # Estimated values for each arm
    counts = np.zeros(num_arms)  # Count of how many times each arm is pulled
    total_reward = 0  # Total reward tracker

    # Initialization: Pull each arm once
    for arm in range(num_arms):
        reward = 1 if np.random.rand() < theta[arm] else 0
        counts[arm] = 1
        Q[arm] = reward
        total_reward += reward

    # Main loop: Epsilon-greedy exploration and exploitation
    for t in range(num_arms, N):
        if np.random.rand() < epsilon:
            # Exploration: choose a random arm
            arm = np.random.randint(num_arms)
        else:
            # Exploitation: choose the arm with the highest estimated value
            arm = np.argmax(Q)

        # Simulate pulling the chosen arm
        reward = 1 if np.random.rand() < theta[arm] else 0

        counts[arm] += 1
        Q[arm] += (1 / counts[arm]) * (reward - Q[arm])

        total_reward += reward
```

```
    return total_reward
```

Implementation of UCB algorithm:

```
def ucb(c, N, theta):
    Q = np.zeros(num_arms)
    counts = np.zeros(num_arms)
    total_reward = 0

    for arm in range(num_arms):
        reward = 1 if np.random.rand() < theta[arm] else 0
        Q[arm] = reward
        counts[arm] = 1
        total_reward += reward

    for t in range(num_arms+1, N+1):
        ucb_values = Q + c * np.sqrt((2*np.log(t))/counts)
        arm = np.argmax(ucb_values)
        reward = 1 if np.random.rand() < theta[arm] else 0
        counts[arm] += 1
        Q[arm] += (1/counts[arm])*(reward - Q[arm])
        total_reward += reward
    return total_reward
```

Implementation of Thompson Sampling algorithm:

```
from scipy.stats import beta

def thompson_sampling(N, theta, alpha_init, beta_init):
    alpha = alpha_init.copy()
    beta_ = beta_init.copy()
    total_reward = 0
    for t in range(N):
        sampled_thetas = [np.random.beta(alpha[j], beta_[j])
                          for j in range(num_arms)]
        arm = np.argmax(sampled_thetas)
        reward = 1 if np.random.rand() < theta[arm] else 0
        total_reward += reward
        alpha[arm] += reward
        beta_[arm] += 1 - reward
    return total_reward
```

Problem 3

The results for the three algorithms with various parameters, averaged over 200 trials and 5000 time slots, are presented below:

ε-Greedy

- $\varepsilon = 0.1$: **3408.44**
- $\varepsilon = 0.5$: **3085.66**
- $\varepsilon = 0.9$: **2748.22**

Upper Confidence Bound (UCB)

- $c = 1$: **3408.32**
- $c = 5$: **2979.74**
- $c = 10$: **2829.24**

Thompson Sampling (TS)

- $(\alpha_1, \beta_1) = (1, 1), (\alpha_2, \beta_2) = (1, 1), (\alpha_3, \beta_3) = (1, 1)$: **3480.75**
- $(\alpha_1, \beta_1) = (601, 401), (\alpha_2, \beta_2) = (401, 601), (\alpha_3, \beta_3) = (2, 3)$: **3492.41**

Problem 4

In this analysis, we compare the performance of three popular multi-armed bandit algorithms: ε-Greedy, Upper Confidence Bound (UCB), and Thompson Sampling (TS). The goal is to compute the gaps between the algorithm outputs (aggregated rewards over $N = 5000$ time slots) and the oracle value, and determine which algorithm performs best.

The theoretical best reward is calculated under the assumption that we know the parameters of the Bernoulli distributions for three arms as follows:

$$\theta_1 = 0.7, \quad \theta_2 = 0.5, \quad \theta_3 = 0.4.$$

Thus, the theoretically maximized expectation is:

$$\max_{I(t),t=1,\ldots,N} \mathbb{E}\left[\sum_{t=1}^{N} r_{I(t)}\right] = N \times \theta_1 = 5000 \times 0.7 = 3500.$$

Gap Calculation

The gap between the algorithm reward and the oracle reward of 3500 is calculated as:

$$\text{Gap} = \text{Oracle Value} - \text{Algorithm Reward}$$

1. ε-Greedy:

ε	Algorithm Reward	Gap
0.1	3408.44	91.56
0.5	3085.66	414.34
0.9	2748.22	751.78

2. Upper Confidence Bound (UCB):

c	Algorithm Reward	Gap
1	3408.32	91.68
5	2979.74	520.26
10	2829.24	670.76

3. Thompson Sampling (TS):

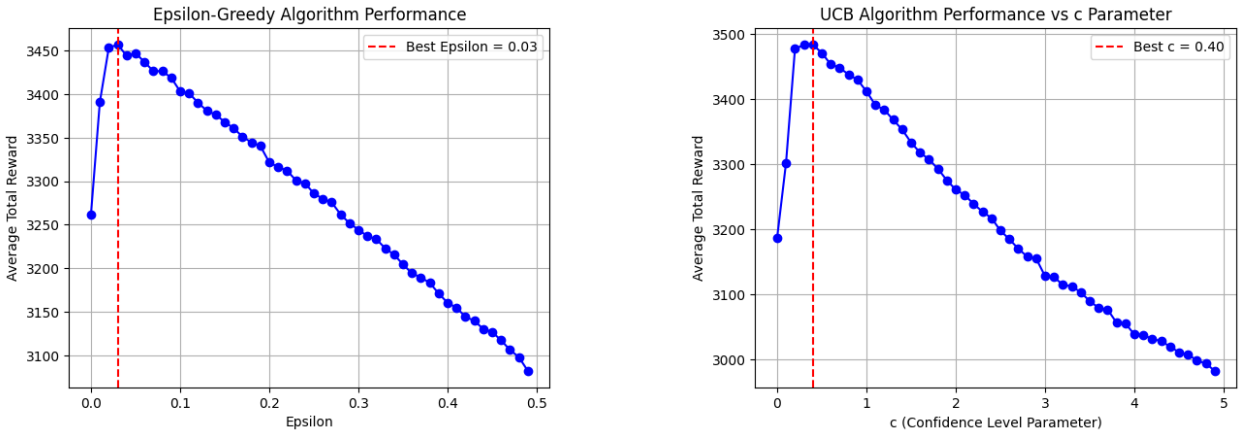
$(\alpha_1, \beta_1), (\alpha_2, \beta_2), (\alpha_3, \beta_3)$	Algorithm Reward	Gap
$(1, 1), (1, 1), (1, 1)$	3480.75	19.25
$(601, 401), (401, 601), (2, 3)$	3492.41	7.59

By optimizing the parameters of each algorithm, we can achieve better performance. After performing the following process for ε-Greedy:

1. Sweep ε from 0 to 0.5 in increments of 0.01.
2. Runs 100 trials for each epsilon value.
3. Averages the total rewards over these 100 trials.
4. Identifies the ε that yields the highest average reward.

we find that the the best ε to maximize the total reward is 0.03, which yields the maximized rewards of 3457.02.

Similarly, for UCB, we sweep the confidence scale c from 0 to 5 in increments of 0.1, and find that the best c is 0.40, which yields the maximized rewards of 3483.90.



(a) ε-Greedy Results (b) UCB Results

Figure 1: Algorithm Performance Analysis

But even with the optimized parameters, the gap of the first two algorithms (42.98 and 16.10 respectively) is still larger than **Thompson Sampling**, which has the smallest gap of 7.59. Therefore, the **Thompson Sampling** algorithm is the best-performing algorithm in this scenario.

Parameter Impact Analysis

1. ε-Greedy Algorithm

   The ε-Greedy algorithm explores with probability ε and exploits with probability $1-\varepsilon$. The parameter ε controls the level of exploration versus exploitation.

   - **Low ε :**
     - Pros: Prefer exploitation to exploration, leading to higher rewards when the current best action is optimal.
     - Cons: Limited exploration may prevent discovery of better actions, especially in complex environments.

- **High ε (e.g., $\varepsilon = 0.5$):**
  - Pros: Increased exploration, potentially leading to the discovery of optimal actions.
  - Cons: Excessive exploration dilutes the focus on known good actions, potentially lowering overall rewards.
- **Optimal ε:** Based on experiments, we find that $\varepsilon = 0.03$ yields the highest average reward of 3457.02, indicating a good balance between exploration and exploitation. And when $\varepsilon > 0.03$, the average reward decreases as the ε (exploration) increases.

2. Upper Confidence Bound (UCB)

   The UCB formula consists of two terms:

   - **Exploitation ($\hat{\theta}(j)$):** This term represents the current mean reward of action j, which is used to exploit the known best action. The algorithm favors actions with a higher expected reward, leading to exploitation.
   - **Exploration ($c \cdot \sqrt{\frac{2\ln(t)}{\mathbf{count}(j)}}$):** This term encourages exploration by adding a bonus to actions that have been chosen fewer times. The bonus is larger for actions that are less tested, which helps to balance exploration with the exploitation of known rewards. The parameter c controls the size of the exploration term. A higher c increases exploration, and a smaller c favors exploitation.

   So the parameter c controls the trade-off between exploration and exploitation in the UCB algorithm, which is discussed as follows:

   - **Low c:**
     - Pros: Promotes exploitation as the confidence bounds become tighter.
     - Cons: Reduced exploration can prevent the algorithm from discovering better actions.
   - **High c:**
     - Pros: Increases exploration by enlarging the confidence bounds, particularly for arms that have been pulled fewer times.
     - Cons: Excessive exploration can reduce the focus on exploitation, potentially leading to suboptimal rewards.
   - **Optimal c:** In our experiments, $c = 0.40$ provides the best performance, yielding a maximum reward of 3483.90. And when $c > 0.40$, the average reward decreases as the c (exploration) increases.

3. Thompson Sampling

   The Thompson Sampling algorithm is a Bayesian method that uses prior and posterior beliefs to update the probability distribution for each action's reward. It assumes a Beta distribution as the prior for the reward probability of each arm, and updates the parameters α_j and β_j based on the observed outcomes.

   Prior Belief:

   - α_j and β_j represent our prior belief about the reward distribution for action j.

- For example, $\alpha_j = 1$ and $\beta_j = 1$ implies that we expect the reward probability for action j to be 50%, but we have low confidence in this belief (uniform distribution).
- A higher value of α_j relative to β_j means that we believe the reward probability for action j is higher, whereas a lower value of α_j relative to β_j suggests that we believe the reward probability is lower.

Effect of α_j and β_j:

- α_j and β_j are updated after each trial, based on whether the reward for action j was successful or not.
- When α_j is large and β_j is small (e.g., $\alpha_j = 2000$, $\beta_j = 8000$), we have a strong belief that the probability of a reward is low (20%), and the algorithm exploits this information.
- When both α_j and β_j are small (e.g., $\alpha_j = 1$, $\beta_j = 1$), there is high uncertainty about the reward distribution, encouraging more exploration of different actions.

Problem 5

The exploration-exploitation trade-off is a central challenge in decision-making processes, particularly in bandit algorithms. It arises when an agent, tasked with maximizing some reward, must decide how much to explore new options (i.e., gather more data) versus exploiting known, high-reward options based on the information it already has.

Exploration involves trying out different actions to gather more information about their outcomes, even if these actions might not seem optimal in the short term. This is essential in the early stages when little is known about the environment.

Exploitation involves choosing the option that has historically provided the highest reward, based on the information the agent has collected. The idea is to maximize immediate rewards using known information.

The trade-off occurs because if the agent always explores, it might fail to capitalize on known high-reward actions. On the other hand, if it always exploits, it risks missing potentially better actions that could be discovered through exploration. Thus, the challenge is in balancing exploration and exploitation over time to optimize overall rewards.

Algorithms for Addressing the Exploration-Exploitation Trade-off

1. Epsilon-Greedy Algorithm: The epsilon-greedy algorithm provides a simple way to balance exploration and exploitation:

   - With probability ε, the agent explores (chooses a random action).
   - With probability $1-\varepsilon$, the agent exploits (chooses the action that has provided the highest reward so far).

   The parameter ε controls the balance between exploration and exploitation. If ε is high, the agent explores more, while if it is low, it exploits more.

   Challenge: The main limitation of the epsilon-greedy approach is that it uses a fixed ε throughout the process. Over time, an agent might need to explore less and exploit more, but a fixed ε might not reflect this need. Adjusting ε over time (e.g., decreasing ε as the agent learns more) can improve performance, where more exploration is done at the beginning and exploitation increases as certainty builds.

2. Upper Confidence Bound (UCB) Algorithm: The UCB algorithm is based on the idea of balancing exploration and exploitation by considering both the average reward of each action and the uncertainty in the estimate of that reward:

   - For each arm (action), UCB calculates an upper bound on the potential reward based on how many times the arm has been selected and the variance in its reward.
   - The agent then selects the arm with the highest upper bound, balancing the need to exploit the best-known action and explore those with high uncertainty.

   The UCB algorithm relies on Hoeffding's inequality to estimate the confidence intervals for each arm's expected reward. The algorithm rewards actions with high uncertainty to ensure that they are explored adequately while still exploiting actions with the highest observed reward.

Advantage: As time progresses, the UCB algorithm places more weight on exploitation as uncertainty decreases, gradually refining the agent's knowledge. It inherently balances exploration and exploitation without requiring manual adjustment of the exploration rate.

3. Thompson Sampling Algorithm Thompson Sampling takes a probabilistic approach to the exploration-exploitation trade-off, using Bayesian inference:

   - Each arm is modeled by a Beta distribution (since Beta is the conjugate prior for Bernoulli/binomial likelihood), and the parameters of this distribution represent the belief about the arm's reward.
   - The agent samples from the Beta distributions and selects the arm with the highest sampled reward.
   - After each trial, the agent updates the Beta distribution based on the observed reward, refining its belief about the arm's expected reward.

   The agent uses prior knowledge (if available) and updates its beliefs about the rewards using the observed data. In this way, the exploration-exploitation trade-off is handled naturally by the sampling process: arms with higher uncertainty (higher variance in their Beta distribution) are explored more, while arms with lower uncertainty are exploited.

   Advantage: Thompson Sampling is highly effective and tends to outperform epsilon-greedy and UCB in many scenarios, particularly when the true reward distributions are well-modeled by Beta distributions. The algorithm's probabilistic nature makes it flexible and robust across different situations.

Ultimately, the exploration-exploitation trade-off is about finding a strategy that maximizes cumulative rewards over time. While epsilon-greedy is easy to implement and useful for simpler environments, UCB and Thompson Sampling are more sophisticated and provide better performance in many complex scenarios, especially when the agent's knowledge about the environment is continuously evolving.

Problem 6

Problem Settings (Dependent Case)

We examine a multi-armed bandit problem featuring three interdependent arms. This scenario introduces a dependency between the arms, which is set as follows:

After each arm pull, the probabilities are adjusted based on the outcome:

- **If a reward is obtained** from pulling arm j (reward $= 1$):

$$\begin{aligned} \theta_j &\leftarrow \max(\theta_j - p, 0) \\ \theta_k &\leftarrow \min(\theta_k + \frac{p}{2}, 1) \quad \forall k \neq j \end{aligned}$$

- **If no reward is obtained** from pulling arm j (reward $= 0$):

$$\begin{aligned} \theta_j &\leftarrow \min(\theta_j + p, 1) \\ \theta_k &\leftarrow \max(\theta_k - \frac{p}{2}, 0) \quad \forall k \neq j \end{aligned}$$

These adjustments ensure that the reward probabilities remain within the valid range $[0, 1]$.

Experimental Setup To evaluate the performance of the algorithms under the independent settings, the following parameters are used:

- **Number of Time Steps** (N): 5000.
- **Number of Trials** (repeat_time): 100.
- **Adjustment Parameter** (p): 0.005.

Objective The primary goal is to determine the optimal algorithmic parameters that maximize the average total reward over N time steps across multiple trials.

Algorithm Design

For the dependent case, we test the three original algorithms on dependent arms. Then, we implement a new algorithm, `Dependency-Aware Thompson Sampling` (DATS), which adapts the Thompson Sampling algorithm to account for the interdependence between arms to obtain a better result.

Algorithm Description: The Dependency-Aware Thompson Sampling with Dynamic Environment Updates algorithm is designed to handle non-stationary environments where arm probabilities change over time and there are potential dependencies between arms. The algorithm operates as follows:

1. **Initialization:**
   - Initialize probabilities (θ) for each arm.

- Set initial Beta distribution parameters (α and β) for each arm.
- Define parameters: N (number of iterations), p (probability update rate), ϵ (exploration rate), γ (dependency factor).

2. **Arm Selection:** For each iteration t from 1 to N:
   - With probability ϵ, explore by choosing a random arm.
   - Otherwise (probability $1 - \epsilon$), exploit using Thompson Sampling:
     - For each arm i, sample a value from Beta(α_i, β_i).
     - Choose the arm with the highest sampled value.
3. **Reward Observation:**
   - Observe a reward (0 or 1) based on the current probability of the chosen arm.
   - Add the reward to the total reward.
4. **Beta Distribution Update:**
   - If reward = 1:
     - Increment α of the chosen arm by 1.
     - If $\gamma > 0$, increment α of all other arms by γ.
   - If reward = 0:
     - Increment β of the chosen arm by 1.
     - If $\gamma > 0$, increment β of all other arms by γ.
5. **Return:** Total accumulated reward over all iterations.

Key Features: This approach combines several strategies to handle the challenges of a non-stationary, dependent arm environment:

- The ϵ-greedy method ensures continued exploration, which is crucial for detecting changes in the environment.
- Thompson Sampling provides a balance between exploration and exploitation based on the current beliefs about arm probabilities.

Algorithm 1 Dependency-Aware Thompson Sampling

Ensure: Total cumulative reward total_reward

$\alpha \leftarrow \alpha_{\text{init}}$
$\beta \leftarrow \beta_{\text{init}}$
$K \leftarrow \text{length}(\alpha)$
$\theta_{\text{current}} \leftarrow \theta$
total_reward $\leftarrow 0$
for $t \leftarrow 1$ to N **do**
    if $\text{Uniform}(0,1) < \epsilon$ **then**
        chosen_arm $\leftarrow \text{Random}(\{1, \ldots, K\})$
    else
        for $i \leftarrow 1$ to K **do**
            $\text{samples}[i] \leftarrow \text{Beta}(\alpha[i], \beta[i])$
        end for
        chosen_arm $\leftarrow \arg\max(\text{samples})$
    end if
    reward $\leftarrow \mathbb{1}[\text{Uniform}(0,1) < \theta_{\text{current}}[\text{chosen\_arm}]]$
    total_reward $\leftarrow$ total_reward + reward
    if reward $= 1$ **then**
        $\alpha[\text{chosen\_arm}] \leftarrow \alpha[\text{chosen\_arm}] + 1$
        for other_arm $\in \{1, \ldots, K\} \setminus \{\text{chosen\_arm}\}$ **do**
            $\alpha[\text{other\_arm}] \leftarrow \alpha[\text{other\_arm}] + \gamma$
        end for
    else
        $\beta[\text{chosen\_arm}] \leftarrow \beta[\text{chosen\_arm}] + 1$
        for other_arm $\in \{1, \ldots, K\} \setminus \{\text{chosen\_arm}\}$ **do**
            $\beta[\text{other\_arm}] \leftarrow \beta[\text{other\_arm}] + \gamma$
        end for
    end if
end for
return total_reward

Results and Analysis

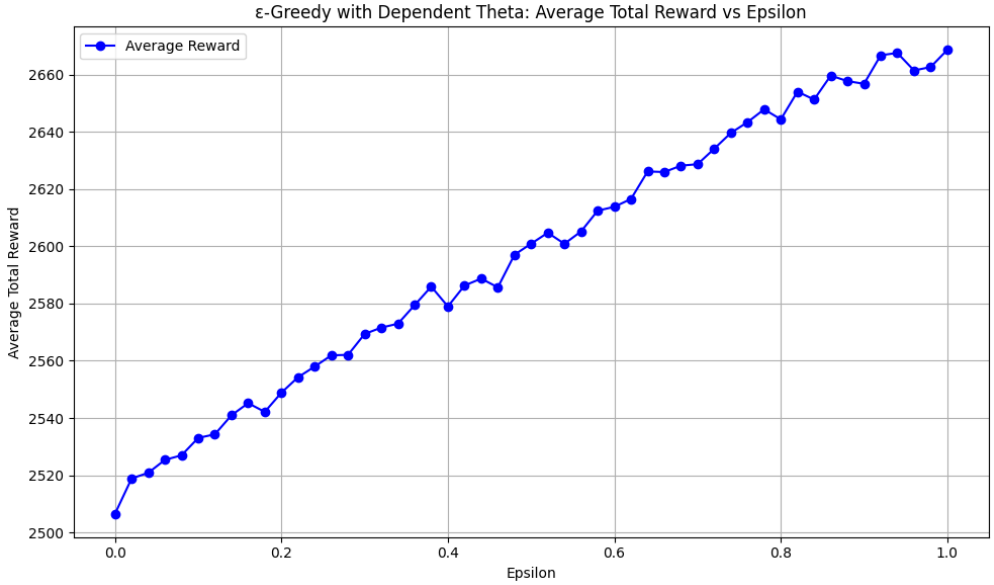


Figure 2: Dependent ε-Greedy Performance

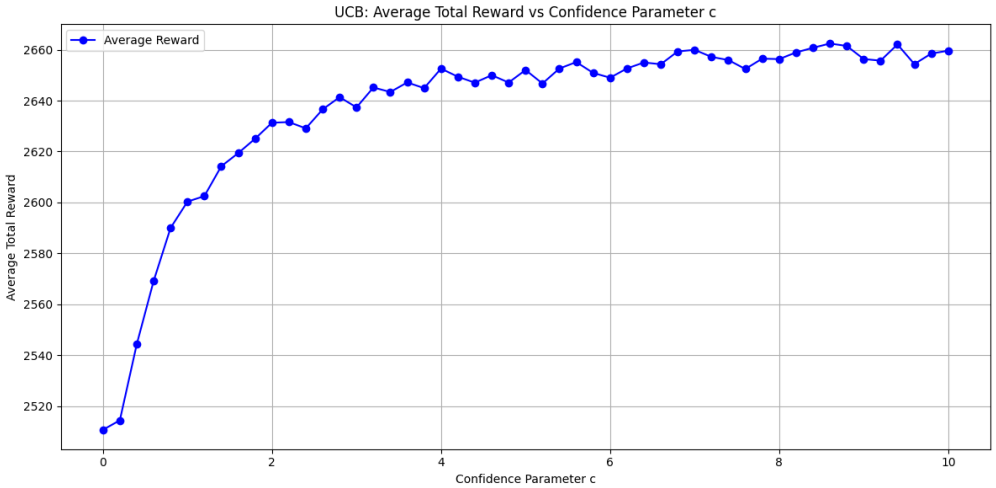


Figure 3: Dependent UCB Performance

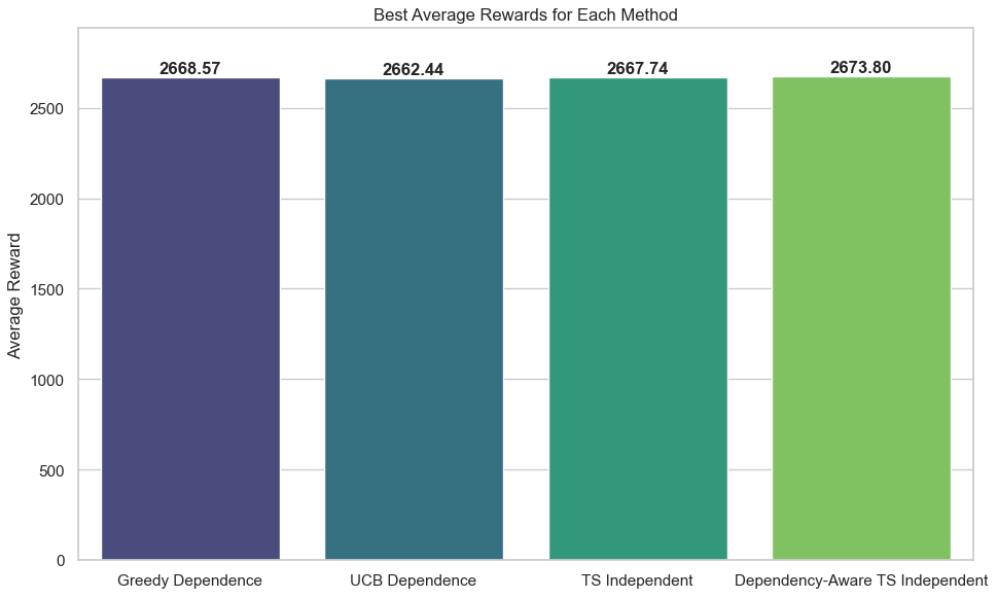


Figure 4: Comparison of TS and DATS with Different Alpha1

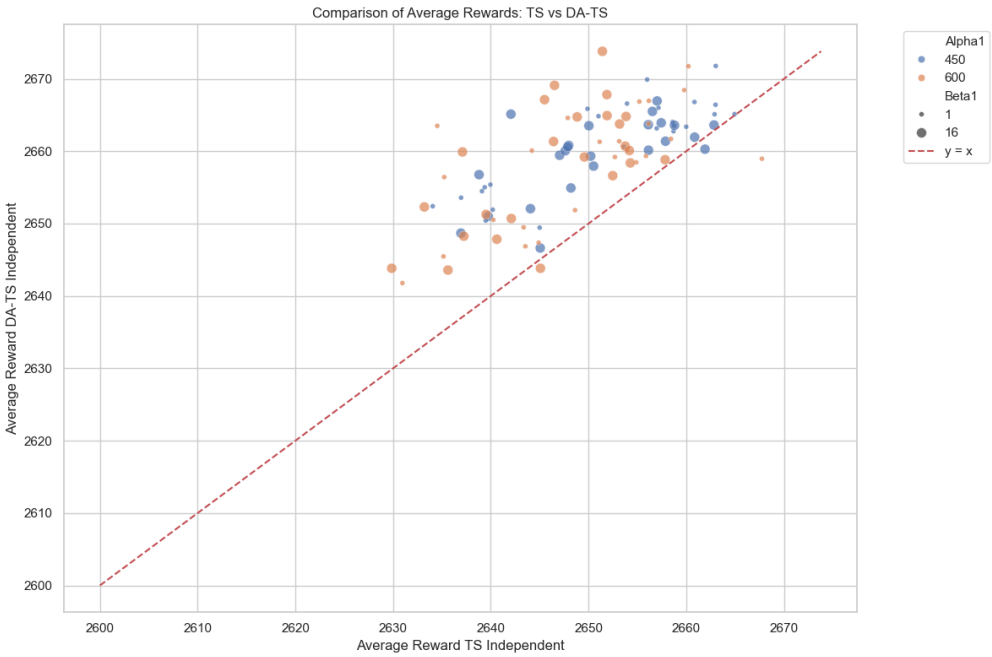


Figure 5: Comparison of TS and DATS

By experimenting, we receive the following results:

Algorithm	**Best Parameters**	**Maximum Reward**
ε-Greedy	$\varepsilon = 1.00$	2668.57
UCB	$c = 8.60$	2662.44
Thompson Sampling	$\alpha = [600, 300, 450]$ $\beta = [1, 1, 31]$	2667.74
Dependency-Aware TS	$\alpha = [600, 450, 450]$ $\beta = [16, 16, 31]$ $\epsilon = 0.001$ $\gamma = 10^{-6}$	2673.80

Table 1: Performance Comparison of Bandit Algorithms

The experimental results presented in Figures 2, 3, 4, 5, and Table 1 provide compelling evidence for the superiority of our proposed Dependency-Aware Thompson Sampling (DATS) algorithm over the other four methods examined: Greedy Dependence, UCB Dependence, and TS Independent.

1. **Superior Performance:** As shown in Figure 4 and Table 1, DATS achieves the highest average reward (2673.80) among all methods tested. This outperforms Greedy Dependence (2668.57), UCB Dependence (2662.44), and TS Independent (2667.74), demonstrating DATS's ability to make more informed decisions in multi-armed bandit problems with dependent arms.

2. **Consistent Outperformance:** The scatter plot in Figure 5 illustrates that DATS consistently outperforms the TS Independent method across various parameter settings. The majority of points lie above the $y = x$ line, indicating that DATS yields higher average rewards in most scenarios.

3. **Robustness to Hyperparameters:** Unlike the Greedy Dependence (Figure 2) and UCB Dependence (Figure 3) methods, which show high sensitivity to their respective hyperparameters (ϵ and confidence parameter c), DATS demonstrates more stable performance across different settings. This robustness is a crucial advantage in real-world applications where optimal hyperparameter tuning may be challenging.

Part II: Bayesian Bandit Algorithms

Problem 1

Experiment Settings

The simulation was conducted with the following parameters:

- **True Success Probabilities:** The true success probabilities for the two arms were set to: $\theta_{\text{true}} = [0.7, 0.5]$
- **Beta Distribution Priors:** The prior parameters for the Beta distributions were initialized as: $\alpha_{\text{prior}} = [1, 1], \quad \beta_{\text{prior}} = [1, 1]$
- **Gamma Values:** The discount factor γ was varied linearly from 0.95 to 1.0 in increments of 0.01, resulting in 50 values: $\gamma \in \text{linspace}(0.95, 1.0, 50)$
- **Number of Time Steps:** Each trial consisted of 5000 time steps: $T = 5000$
- **Number of Trials per Gamma:** The experiment was repeated 50 times for each γ value: Repeats $= 50$

Results

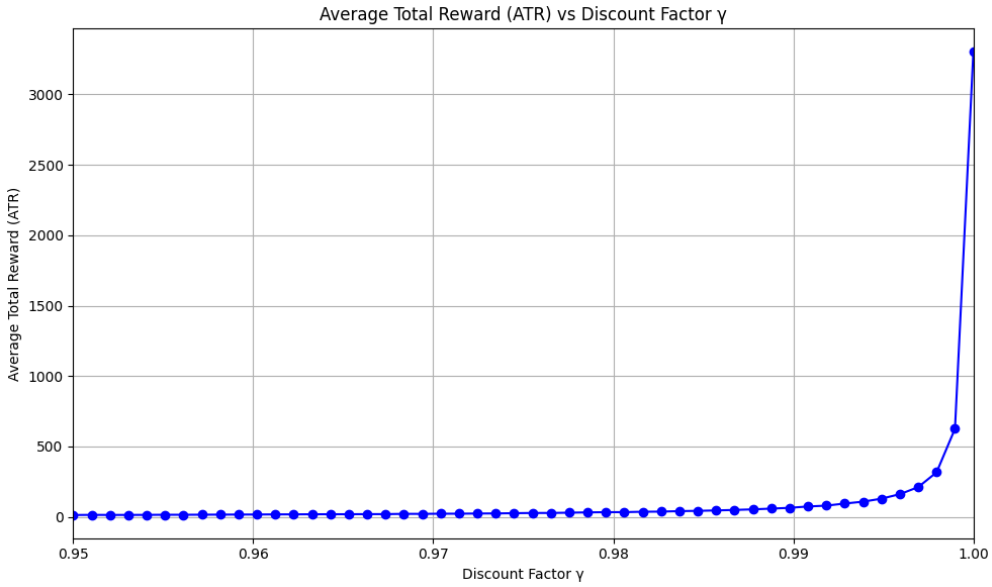


Figure 6: Intuitive Outcomes with Different γ Values

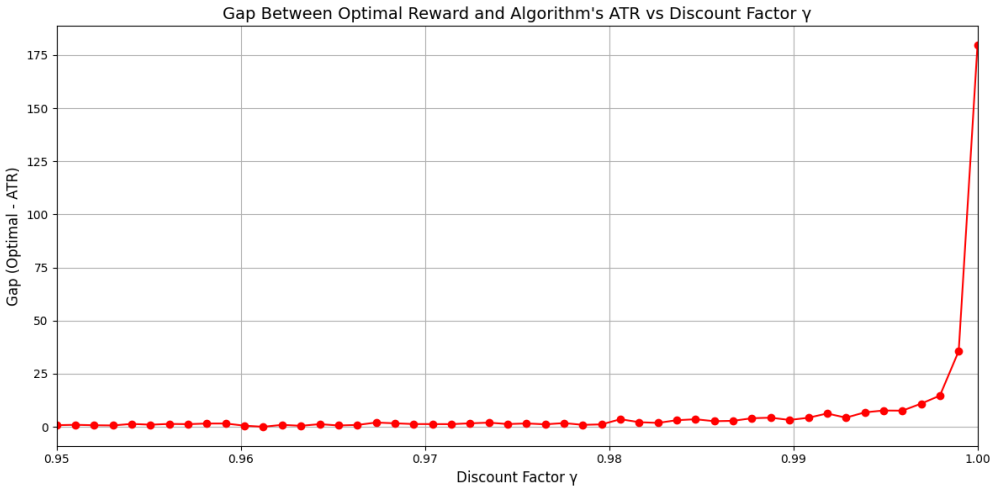


Figure 7: Gaps Between Optimal and Intuitive Algorithm Outcomes with Different γ Values

In this part, we implemented an intuitive algorithm and calculated the theoretical optimal rewards.

The theoretical optimal reward is calculated as:

$$\text{Optimal Reward} = \theta_{\text{best}} \times \frac{1 - \gamma^{\text{time\_steps}}}{1 - \gamma}$$

where θ_{best} is the highest success probability among the arms, γ is the discount factor, and *time_steps* is the total number of pulls in a trial.

The results of our experiments are visualized in the two figures:

From Figure 7, we observe that the gap between the optimal reward and the Average Total Reward (ATR) produced by the intuitive algorithm remains small for most values of the discount factor γ. This demonstrates that the intuitive algorithm behaves very well in most cases.

Additionally, as shown in Figure 6, the ATR increases as γ approaches 1.0, which aligns with our expectations since the algorithm becomes more conservative, favoring long-term rewards.

Problem 2

Situation when the intuitive algorithm fails to perform optimally

We set the theta values for the two arms as $\theta_{\text{true}} = [0.3, 0.6]$. By running the experiment with the same other settings as in Problem 1, we get the following results:

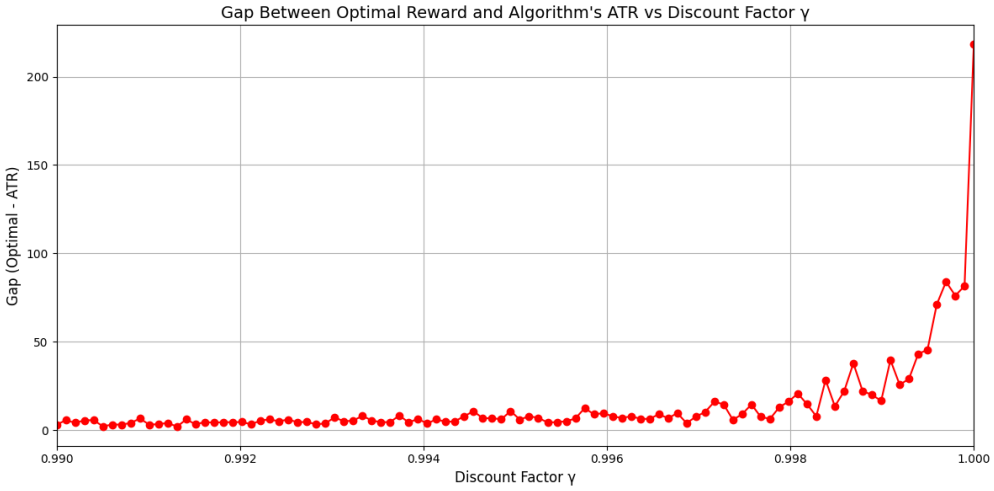


Figure 8: Gaps Between Optimal and Intuitive Algorithm Outcomes with Different γ Values

Since the $\gamma \in [0, 0.99]$ shows no significant difference in the gaps between the optimal and intuitive algorithm outcomes, we simply omit testing these values.

As we can see in the Figure 8, the gap between the optimal reward and the Average Total Reward (ATR) produced by the intuitive algorithm is significantly larger when $\gamma \in [0.998, 1.000]$. This indicates that the intuitive algorithm fails to perform optimally in this scenario.

Comparison between the intuitive algorithm and TS

We compare the performance of the intuitive algorithm with that of Thompson Sampling (TS) by plotting the Average Total Reward (ATR) for both algorithms across different values of the discount factor γ. The settings are the same as above. We get the following results:

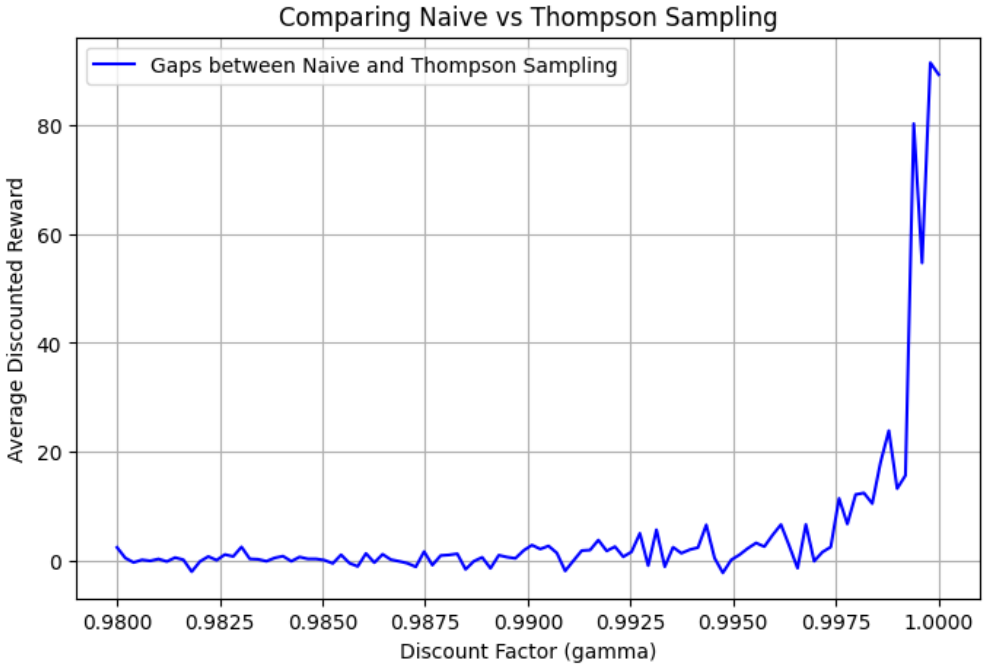


Figure 9: Comparison of Intuitive Algorithm and Thompson Sampling

As shown in Figure 9, the Average Total Reward (ATR) produced by Thompson Sampling (TS) is significantly higher than that of the intuitive algorithm across $\gamma \in [0.9875, 1.0000]$. This indicates that Thompson Sampling outperforms the intuitive algorithm in this scenario.

Therefore, by these two experiments, we can conclude that the intuitive algorithm may fail to perform optimally in certain situations, and Thompson Sampling (TS) can provide better results in such cases.

Problem 3

Problem Statement

For the expected total reward under an optimal policy, show that the following recurrence equation holds:

$$R_1(\alpha_1, \beta_1) = \frac{\alpha_1}{\alpha_1 + \beta_1}[1 + \gamma R(\alpha_1 + 1, \beta_1, \alpha_2, \beta_2)] + \frac{\beta_1}{\alpha_1 + \beta_1}[\gamma R(\alpha_1, \beta_1 + 1, \alpha_2, \beta_2)];$$
$$R_2(\alpha_2, \beta_2) = \frac{\alpha_2}{\alpha_2 + \beta_2}[1 + \gamma R(\alpha_1, \beta_1, \alpha_2 + 1, \beta_2)] + \frac{\beta_2}{\alpha_2 + \beta_2}[\gamma R(\alpha_1, \beta_1, \alpha_2, \beta_2 + 1)];$$
$$R(\alpha_1, \beta_1, \alpha_2, \beta_2) = \max\{R_1(\alpha_1, \beta_1), R_2(\alpha_2, \beta_2)\}.$$

Proof

At time $t = 0$, the parameters θ_1 and θ_2 are assumed to follow independent Beta distributions with parameters (α_1, β_1) and (α_2, β_2), respectively.

Pull the First Arm

When arm 1 is pulled at time t, the reward is determined by the Bernoulli distribution $\text{Bern}(\theta_1)$:

- With probability θ_1, a success occurs, yielding an immediate reward of 1 and resulting in a posterior distribution $\text{Beta}(\alpha_1 + 1, \beta_1)$. The future reward is discounted by γ, leading to a total future reward of $\gamma R(\alpha_1 + 1, \beta_1, \alpha_2, \beta_2)$.
- With probability $1 - \theta_1$, a failure occurs, yielding an immediate reward of 0 and resulting in a posterior distribution $\text{Beta}(\alpha_1, \beta_1 + 1)$. The total future reward in this case is $\gamma R(\alpha_1, \beta_1 + 1, \alpha_2, \beta_2)$.

Combining these outcomes, the expected reward from pulling arm 1 is:

$$R_1(\alpha_1, \beta_1) = \theta_1[1 + \gamma R(\alpha_1 + 1, \beta_1, \alpha_2, \beta_2)] + (1 - \theta_1)[\gamma R(\alpha_1, \beta_1 + 1, \alpha_2, \beta_2)].$$

Using the expectation of θ_1 under its Beta distribution, where:

$$E[\theta_1] = \frac{\alpha_1}{\alpha_1 + \beta_1}, \quad E[1 - \theta_1] = \frac{\beta_1}{\alpha_1 + \beta_1},$$

we can rewrite $R_1(\alpha_1, \beta_1)$ as:

$$R_1(\alpha_1, \beta_1) = \frac{\alpha_1}{\alpha_1 + \beta_1}[1 + \gamma R(\alpha_1 + 1, \beta_1, \alpha_2, \beta_2)] + \frac{\beta_1}{\alpha_1 + \beta_1}[\gamma R(\alpha_1, \beta_1 + 1, \alpha_2, \beta_2)].$$

Pull the Second Arm

Similarly, when arm 2 is pulled, the reward is determined by the Bernoulli distribution $\text{Bern}(\theta_2)$. The outcomes are:

- With probability θ_2, a success occurs, yielding an immediate reward of 1 and resulting in a posterior distribution Beta$(\alpha_2 + 1, \beta_2)$. The total future reward is $\gamma R(\alpha_1, \beta_1, \alpha_2 + 1, \beta_2)$.
- With probability $1 - \theta_2$, a failure occurs, yielding an immediate reward of 0 and resulting in a posterior distribution Beta$(\alpha_2, \beta_2 + 1)$. The total future reward in this case is $\gamma R(\alpha_1, \beta_1, \alpha_2, \beta_2 + 1)$.

Combining these outcomes, the expected reward from pulling arm 2 is:

$$R_2(\alpha_2, \beta_2) = \frac{\alpha_2}{\alpha_2 + \beta_2}[1 + \gamma R(\alpha_1, \beta_1, \alpha_2 + 1, \beta_2)] + \frac{\beta_2}{\alpha_2 + \beta_2}[\gamma R(\alpha_1, \beta_1, \alpha_2, \beta_2 + 1)].$$

The expected total reward under the optimal policy is the maximum of the rewards from pulling either arm 1 or arm 2:

$$R(\alpha_1, \beta_1, \alpha_2, \beta_2) = \max\{R_1(\alpha_1, \beta_1), R_2(\alpha_2, \beta_2)\}.$$

Combining all results, we have proved that the recurrence equations are:

$$\begin{aligned}
R_1(\alpha_1, \beta_1) &= \frac{\alpha_1}{\alpha_1 + \beta_1}[1 + \gamma R(\alpha_1 + 1, \beta_1, \alpha_2, \beta_2)] + \frac{\beta_1}{\alpha_1 + \beta_1}[\gamma R(\alpha_1, \beta_1 + 1, \alpha_2, \beta_2)], \\
R_2(\alpha_2, \beta_2) &= \frac{\alpha_2}{\alpha_2 + \beta_2}[1 + \gamma R(\alpha_1, \beta_1, \alpha_2 + 1, \beta_2)] + \frac{\beta_2}{\alpha_2 + \beta_2}[\gamma R(\alpha_1, \beta_1, \alpha_2, \beta_2 + 1)], \\
R(\alpha_1, \beta_1, \alpha_2, \beta_2) &= \max\{R_1(\alpha_1, \beta_1), R_2(\alpha_2, \beta_2)\}.
\end{aligned}$$

Problem 4

Problem Statement

How to solve the recurrence equations exactly or approximately?

Solution Approach

Dynamic Programming (DP) is a powerful method that, in theory, can provide exact solutions to the optimal policy. The principle of optimality, which underlies DP, ensures that if we can evaluate all possible states, we will find the globally optimal solution. However, the transition from theory to practice introduces several challenges that necessitate an approximate solution:

Infinite State Space The Beta-Bernoulli bandit has an infinite state space. Each arm's Beta distribution parameters (α and β) can grow indefinitely as we observe more outcomes. In theory, DP would require us to compute and store values for every possible combination of $(\alpha_1, \beta_1, \alpha_2, \beta_2)$, which is infeasible.

Computational Constraints Even if we could store an infinite number of states, computing the optimal value for each state would require an infinite number of operations. Real-world computers have finite processing capabilities, making exact computation impossible.

Memory Limitations Storing values for an infinite number of states would require infinite memory, which is not available in practice.

To address these challenges, we employ an approximation strategy:

1. **State Space Truncation:** We introduce a maximum value N_{max} (denoted as M in our algorithm) for each parameter. This effectively "truncates" our infinite state space to a finite one.

2. **Boundary Conditions:** We define boundary conditions for our truncated space (e.g., setting the value to 0 when $\alpha_i + \beta_i = N_{max}$ for either arm).

3. **Value Iteration:** We use iterative updates to approximate the value function, stopping when changes become smaller than a predefined tolerance or after a maximum number of iterations.

This approach allows us to find an approximate solution that is computationally feasible. The quality of this approximation depends on several factors:

- The choice of N_{max}: Larger values allow for a more accurate approximation but increase computational cost.

- The convergence tolerance: Smaller tolerances can provide more accurate results but may require more iterations.
- The discount factor γ: Values closer to 1 consider long-term rewards more heavily but may slow convergence.

In practice, these parameters are often tuned to balance solution quality with computational efficiency. While we sacrifice theoretical exactness, this approach often provides solutions that are "good enough" for practical applications, capturing the essential behavior of the optimal policy within a tractable computation framework.

Pseudocode

The following algorithm implements our solution approach:

Algorithm 2 Solve 2-Armed Beta-Bernoulli Bandit

procedure SOLVE_2ARMED_BANDIT_DP(M, γ, tol, max_iter)
  Input:
    M: Truncation level for α_i, β_i
    γ: Discount factor ($0 < \gamma < 1$)
    tol: Convergence tolerance
    max_iter: Maximum number of iterations
  Initialize R and $policy$ as 4D arrays of size $(M+1) \times (M+1) \times (M+1) \times (M+1)$
  for $it = 1$ to max_iter **do**
    $delta \leftarrow 0$ ▷ Track maximum change in this iteration
    for $\alpha_1 = 1$ to M **do**
      for $\beta_1 = 1$ to M **do**
        for $\alpha_2 = 1$ to M **do**
          for $\beta_2 = 1$ to M **do**
            Compute $R_1(\alpha_1, \beta_1)$ and $R_2(\alpha_2, \beta_2)$ using recurrence relations
            $new\_val \leftarrow \max(R_1, R_2)$
            $old\_val \leftarrow R[\alpha_1, \beta_1, \alpha_2, \beta_2]$
            $R[\alpha_1, \beta_1, \alpha_2, \beta_2] \leftarrow new\_val$
            $policy[\alpha_1, \beta_1, \alpha_2, \beta_2] \leftarrow \arg\max(R_1, R_2)$
            $delta \leftarrow \max(delta, |new\_val - old\_val|)$
          end for
        end for
      end for
    end for
    if $delta < tol$ **then**
      break ▷ Convergence achieved
    end if
  end for
  return $R, policy$
end procedure

This algorithm iteratively computes the value function R and the optimal policy. The policy array stores the optimal action (0 for arm 1, 1 for arm 2) for each state. The algorithm terminates when either the solution converges (change in values less than tol) or the maximum number of iterations is reached.

Problem 5

In this problem, we implement the dynamic programming algorithm to solve the 2-armed Beta-Bernoulli bandit problem with the given recurrence relations. We set the parameters as follows:

1. Discount factors (γ): 100 evenly spaced values from 0.9 to 1
   The experiment tests a range of discount factors to explore their impact on performance.

2. Truncation level for Dynamic Programming (M): 17
   This parameter limits the state space for the DP algorithm, balancing computational feasibility with solution accuracy.

3. Tolerance (tol): 10^{-8}
   The algorithm is considered converged when the maximum change in the value function between iterations falls below this threshold.

4. Maximum iterations (max_iter): 50
   This sets an upper limit on the number of iterations for the DP algorithm, ensuring termination even if the tolerance-based convergence is not reached.

The experiment compares two algorithms:

Algorithm	**Description**
4D Dynamic Programming	Solves the problem exactly (up to the truncation level)
Thompson Sampling	Approximate method using Beta distributions

Table 2: Comparison of algorithms used in the experiment

For each γ value, both algorithms are evaluated over multiple trials. The performance metric is the discounted cumulative reward, defined as:

$$R_{\text{total}} = \sum_{t=1}^{T} \gamma^{t-1} r_t$$

where r_t is the reward at time step t, and T is the total number of time steps (5000 in this experiment).

The results are visualized to compare the algorithms' performance across different discount factors and to analyze the performance gap between them.

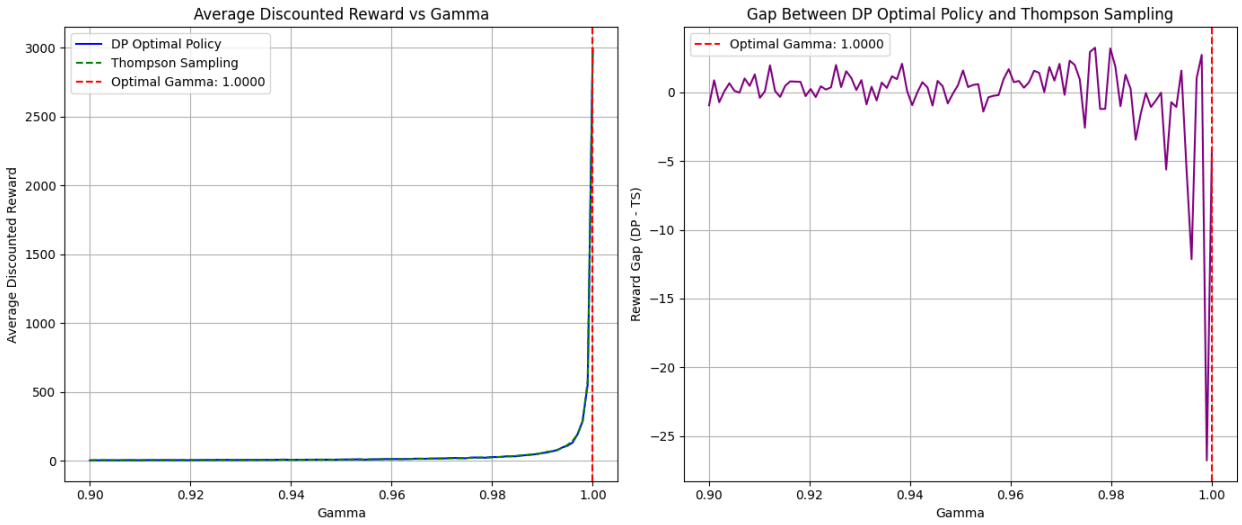


Figure 10: Dynamic Programming vs. Thompson Sampling: Optimal Performance

From Figure 10, we observe that the Dynamic Programming (DP) algorithm outperforms Thompson Sampling across various discount factors (γ), despite that the gap shows a sharp decrease as γ approaches 1. Interestingly, the optimal γ appears to be exactly 1.0000, suggesting that in this scenario, fully prioritizing long-term rewards yields the best performance. The fluctuations in the performance gap for high γ values (0.98-1.00) reveal complex dynamics in the relative efficacy of these algorithms as the planning horizon extends, warranting further investigation into the underlying mechanisms driving these differences.

Conclusion

This study provides a comprehensive evaluation of various bandit learning algorithms, offering valuable insights into their performance characteristics and the nuances of the exploration-exploitation trade-off. Our investigation encompassed both classical and Bayesian approaches, revealing important findings that contribute to the broader understanding of sequential decision-making under uncertainty.

In Part I, we examined the performance of ε-greedy, Upper Confidence Bound (UCB), and Thompson Sampling (TS) algorithms in a classical multi-armed bandit setting. Our results demonstrated that Thompson Sampling consistently outperformed the other algorithms across various parameter settings, achieving the smallest gap from the oracle value. This superior performance underscores the effectiveness of probabilistic methods in balancing exploration and exploitation.

The introduction of arm dependencies in our experiments highlighted the adaptability of these algorithms to more complex environments. Notably, our proposed Dependency-Aware Thompson Sampling (DATS) algorithm showed improved performance in this setting, illustrating the potential for tailored approaches in specific problem domains.

Part II of our study delved into Bayesian bandit algorithms, focusing on discounted reward scenarios. We derived and implemented a dynamic programming solution for the optimal policy, providing a benchmark for comparison with more computationally efficient heuristics. The results revealed that while intuitive algorithms perform well in many cases, they can fail to achieve optimality under certain conditions, particularly as the discount factor approaches 1.

Our analysis of the recurrence equations for the expected total reward under an optimal policy offers theoretical insights into the structure of the problem. The approximate solution method we developed, using state space truncation and value iteration, provides a practical approach to solving these complex problems within computational constraints.

The comparative analysis between the dynamic programming solution and Thompson Sampling across different discount factors yielded intriguing results. The DP approach consistently outperformed TS, the performance gap narrowed significantly (though TS outperforms DP significantly as γ is very close to 1), suggesting that simpler heuristics may be nearly optimal.

These findings have important implications for real-world applications of bandit algorithms. The superior performance of Thompson Sampling in various settings suggests its potential as a robust default choice for many problems. However, the success of the DATS algorithm in dependent arm scenarios highlights the value of domain-specific adaptations. Furthermore, the near-optimality of heuristic methods indicates that computationally intensive exact solutions may not always be necessary in practice.

Future research could explore several promising directions:

- Developing more sophisticated dependency models and corresponding algorithms to handle complex real-world scenarios.

- Investigating the theoretical properties of the Dependency-Aware Thompson Sampling algorithm and deriving bounds on its regret.
- Improve the accuracy and efficiency of the DP algorithm under the circumstances where the discount factor approaches 1.
- Exploring the application of these algorithms to specific domains such as online advertising, clinical trials, or recommendation systems.

In conclusion, this study advances our understanding of bandit algorithms' behavior across various settings and parameter ranges. By rigorously comparing classical and Bayesian approaches, we have provided insights that can guide algorithm selection and design in practical applications. As sequential decision-making problems continue to grow in importance across numerous fields, the insights gained from this work contribute to the ongoing development of efficient and effective reinforcement learning strategies.

Contributions of Team Members

- **Vuong An Thuy:** Part I, Part II Problem 4 & 5, LaTeX formatting, code formatting, and report writing.
- **Luc Trieu:** Part II
- **Pham Canh Nhien:** Emotional Value