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

Jingran Fan Anrui Wang Zhao Lu

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):

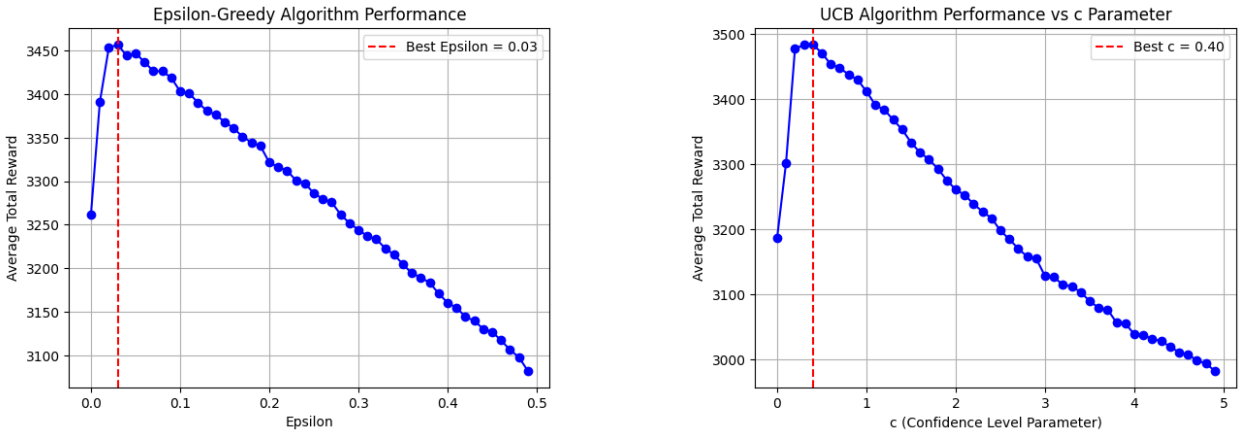
$(\alpha_1, \beta_1), (\alpha_2, \beta_2), (\alpha_3, \beta_3)$	Algorithm Reward	Gap
$(1, 1), (1, 1), (1, 1)$	3480.75	19.25
$(601, 401), (401, 601), (2, 3)$	3492.41	7.59

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
- 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)

- There are 3 arms, each arm $j \in \{1, 2, 3\}$, and each arm j has a reward distribution following a Bernoulli distribution with mean θ_j, i.e., the probability of getting a reward of 1 is θ_j, and the probability of getting a reward of 0 is $1 - \theta_j$.
- In the dependent case, the rewards of the arms are not independent. Specifically, the reward of one arm may influence or be influenced by the rewards of other arms. Assume that the three arms follows linear dependency:

  $$\theta_1 = \alpha \cdot \theta_2 + \beta \cdot \theta_3 + \gamma$$

  where α, β, and γ are constants that define the dependency structure. This could indicate that if arm 2 performs well, arm 1 is more likely to perform well, and vice versa.
- In order to make the tests easier, we assume that the equation is

  $$\theta_1 = 0.5 \cdot \theta_2 + 0.5 \cdot \theta_3$$

  This is valid because we ensure that θ_1, θ_2, and θ_3 are all in the range of $[0, 1]$.

Algorithm Design

In order to obtain a better result, we design four algorithms: `greedy_dependence`, `ucb_dependence`, `thompson_sampling_dependence`, and `dependency_aware_thompson_sampling`.

The first three use the same method as before. We just simply make probability of the arm change.

The fourth algorithm initializes Beta distributions for each arm to model reward uncertainties. In each of N rounds, it samples from these distributions and selects the arm with the highest sample. After observing the reward, it updates the chosen arm's Beta parameters. Crucially, if arm 1 (the potentially dependent arm) is selected, it also updates arms 2 and 3's parameters by a fraction λ of the reward. This approach allows the algorithm to learn and exploit potential dependencies without prior knowledge of their exact nature. The algorithm balances exploration and exploitation, aiming to maximize total reward over time by adapting its arm selection strategy based on observed outcomes and possible inter-arm dependencies.

Algorithm 1 Revised Dependency-Aware Thompson Sampling

Require: Number of time steps N, Initial Beta parameters $\alpha_{\text{init}} = [\alpha_1, \alpha_2, \alpha_3]$, $\beta_{\text{init}} = [\beta_1, \beta_2, \beta_3]$

Initialize $\alpha \leftarrow \alpha_{\text{init}}$
Initialize $\beta \leftarrow \beta_{\text{init}}$
Initialize total_reward $\leftarrow 0$
for $t = 1$ to N **do**
    for $k = 1$ to 3 **do**
        Sample $\tilde{\theta}_k \sim \text{Beta}(\alpha_k, \beta_k)$
    end for
    Select arm $j \leftarrow \arg\max_{k \in \{1,2,3\}} \tilde{\theta}_k$
    Observe reward $r \sim \text{Bernoulli}(\theta_j)$
    total_reward $\leftarrow$ total_reward $+ r$
    $\alpha_j \leftarrow \alpha_j + r$
    $\beta_j \leftarrow \beta_j + (1 - r)$
    if $j = 1$ **then**
        $\alpha_2 \leftarrow \alpha_2 + \lambda r$
        $\beta_2 \leftarrow \beta_2 + \lambda(1 - r)$
        $\alpha_3 \leftarrow \alpha_3 + \lambda r$
        $\beta_3 \leftarrow \beta_3 + \lambda(1 - r)$
    end if
end for
return total_reward

Results and Analysis

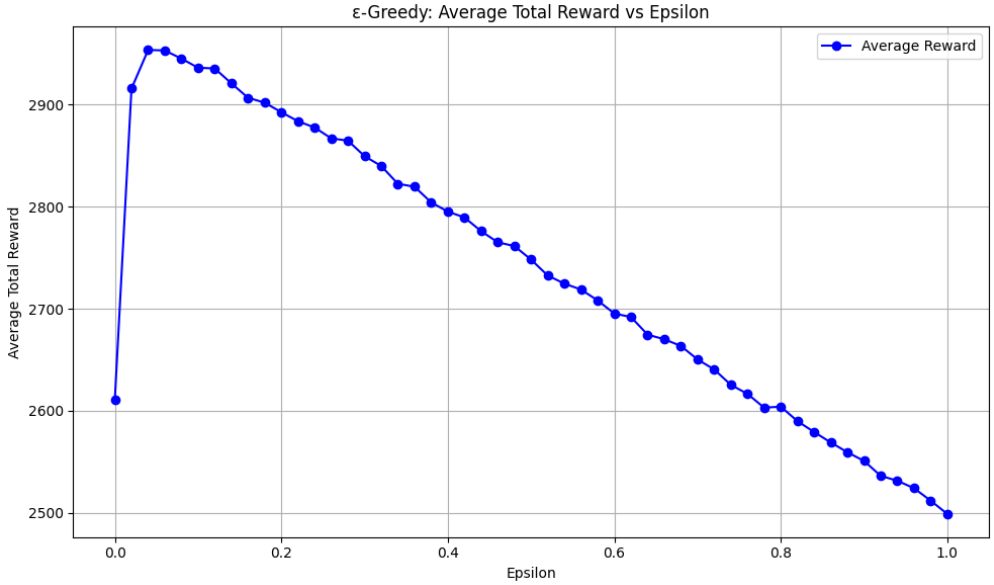


Figure 2: Dependent ε-Greedy Performance

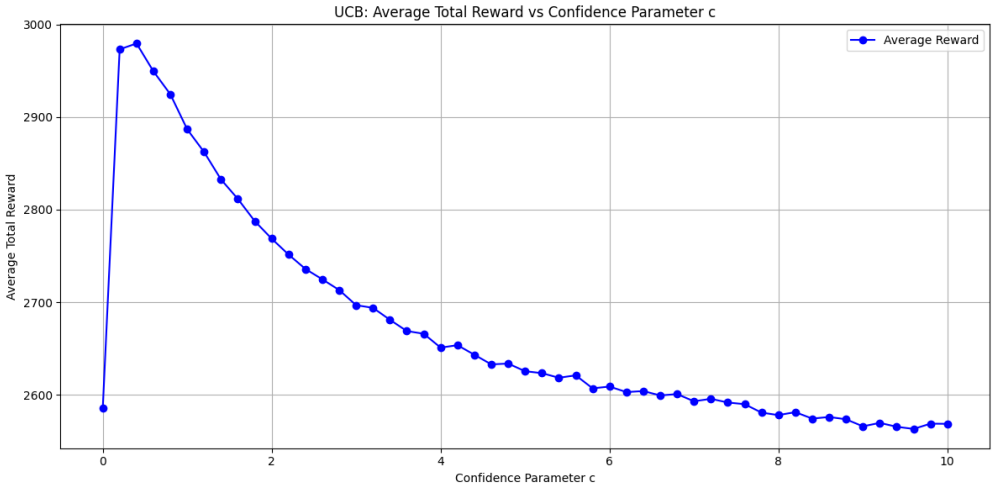


Figure 3: Dependent UCB Performance

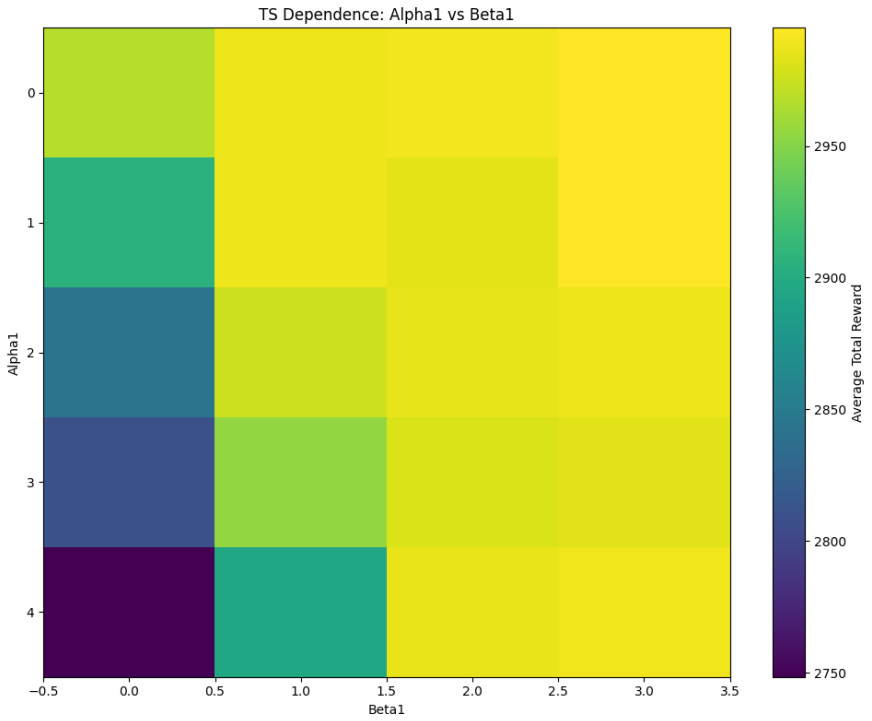


Figure 4: Standard Thompson Sampling Performance

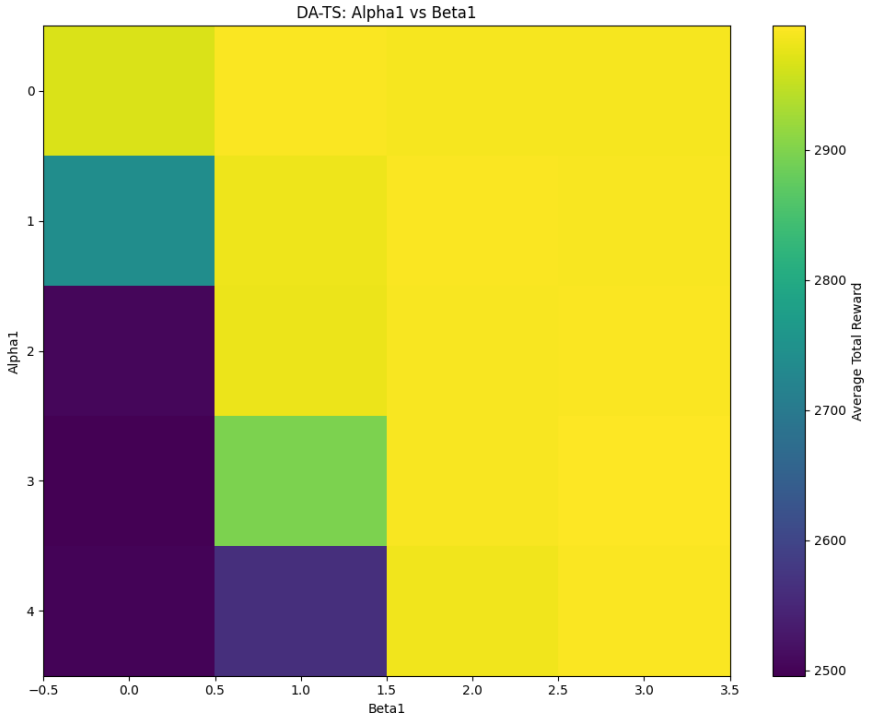


Figure 5: Initial Analysis of Dependency-Aware Thompson Sampling

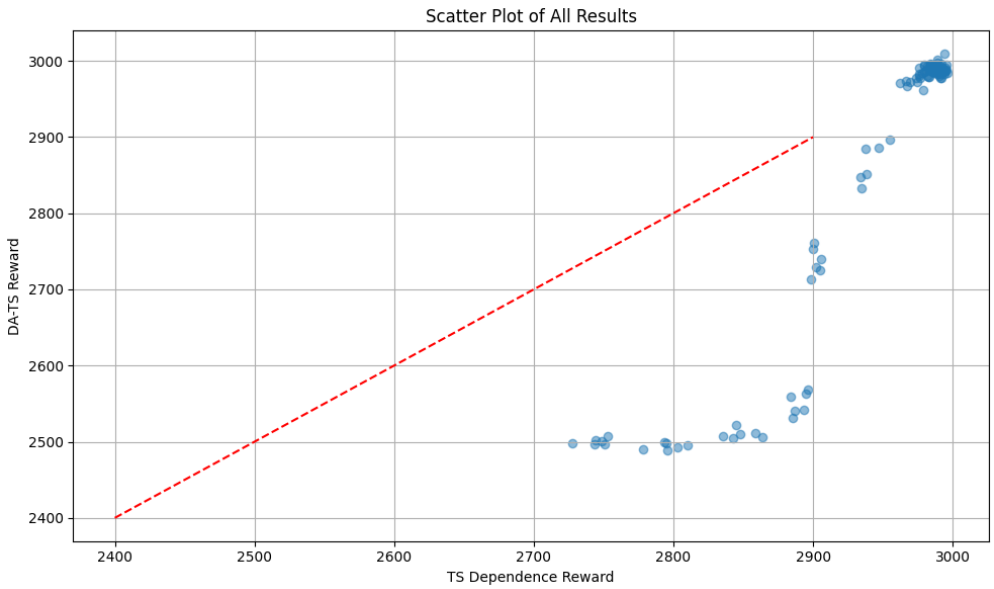


Figure 6: Comparison of TS and DATS

By experimenting, we receive the following results:

Algorithm	**Best Parameters**	**Maximum Reward**
ε-Greedy	$\varepsilon = 0.04$	2953.50
UCB	$c = 0.40$	2979.58
Thompson Sampling	$\alpha = [201, 1]$ $\beta = [601, 3]$	2996.08
Dependency-Aware TS	$\alpha = [1, 2]$ $\beta = [201, 3]$	3009.12

Table 1: Performance Comparison of Bandit Algorithms

According to Table 1, Dependency-Aware Thompson Sampling (DA-TS) achieved the highest maximum reward (3009.12) among all algorithms tested. This represents a significant improvement over other algorithms.

The performance analysis reveals distinct patterns in the heatmaps of standard Thompson Sampling (Figure 4) and Dependency-Aware Thompson Sampling (Figure 5). While TS demonstrates better performance with lower α values, DA-TS performs optimally with higher α and β values. The comparative analysis (Figure 6) demonstrates that DA-TS consistently outperforms standard TS across different parameter settings, indicating its superior ability to adapt to the underlying dependency structure of the rewards.

Part II: Bayesian Bandit Algorithms