Programming assignment 3: Group 9

Part 1 / Algorithm A -

Problem definition:

The problem described for Algorithm A involves a video recommendation system with users receiving recommendations from one of K types of videos, where a user either clicks on the video or ignores it. The system aims to maximize its revenue from N users, but it doesn't know the probabilities of pk (click rates) and ak (revenue distributions). The system must choose the video class that maximizes the expected revenue E(pkRk), but since these parameters are unknown, the algorithm needs to adopt a learning approach.

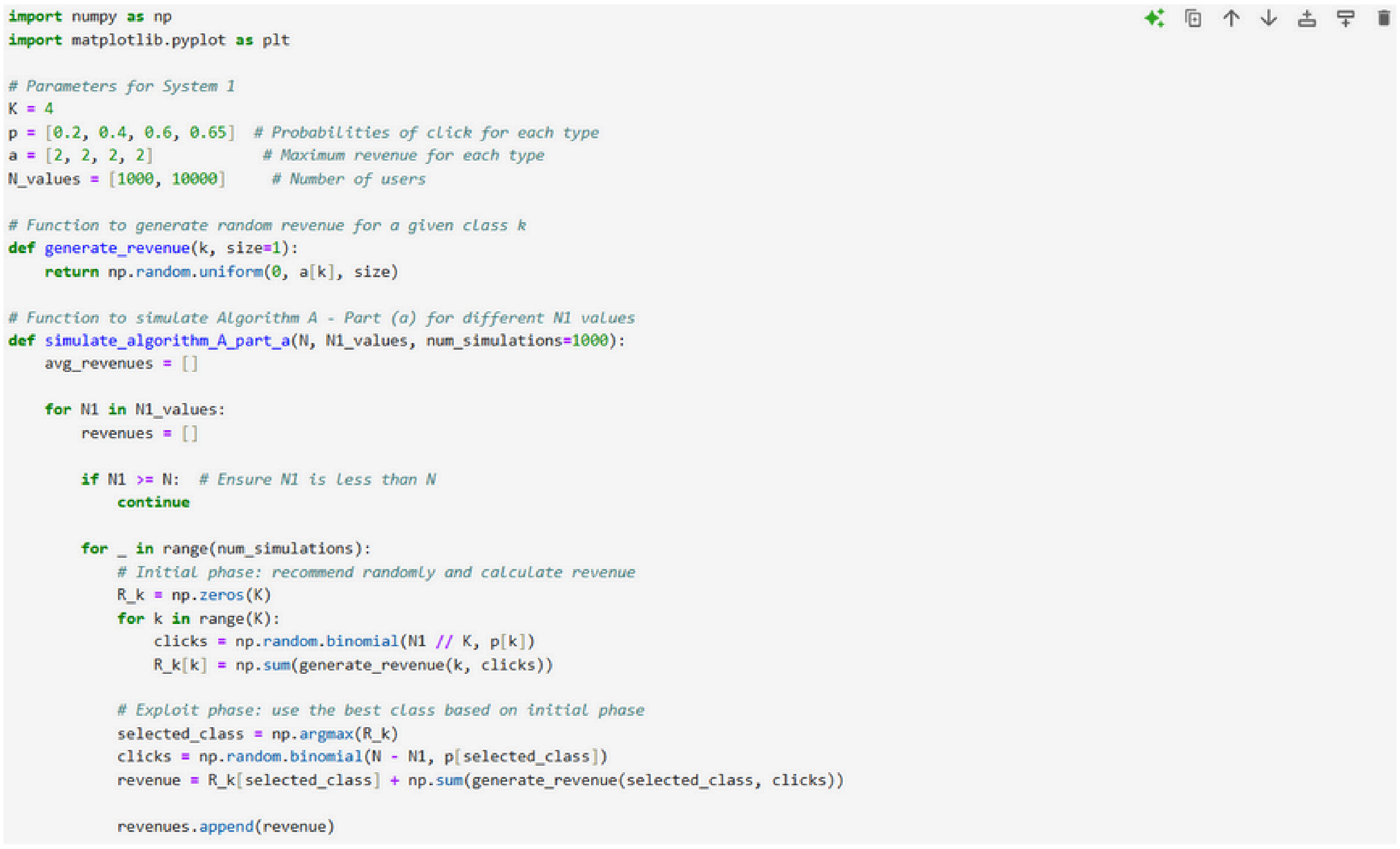
Approach for Algorithm A:

1. **Initial Sampling Phase (N1 Users)**: The system initially selects N1 users to randomly recommend videos from each K class, allocating them N1/K users each. The revenue Rk for each class is recorded during this phase.
2. **Exploit Phase (Remaining Users)**: For the remaining N−N1 users, the system recommends videos from the class that yielded the highest revenue in the initial phase.
3. The challenge lies in finding the optimal N1 value:
    - **If N1 is too small**, the system may not have enough data, increasing the likelihood of choosing a suboptimal class. Intuitively, the wrong type may be selected with a higher probability.
    - **If N1 is too large**, insu cient users will be left to benefit from the collected data. Also, there is not enough time to use the more reliable information collected from the first N1 users.

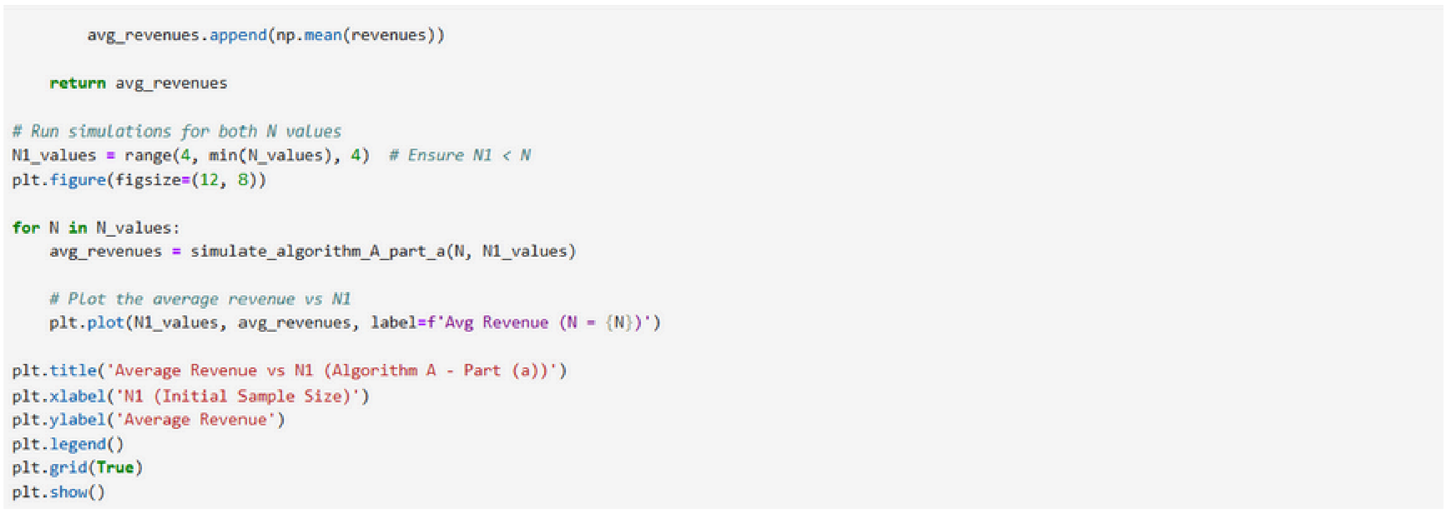
Tasks:

- Simulate the algorithm for various N1 values and compute the sample average revenue.
- Plot the revenue R(N1) against N1 to identify the optimum.
- Compute the theoretical probability of picking the wrong type after N1 users, assuming pk and ak are known, and compare it to empirical results.

Code:

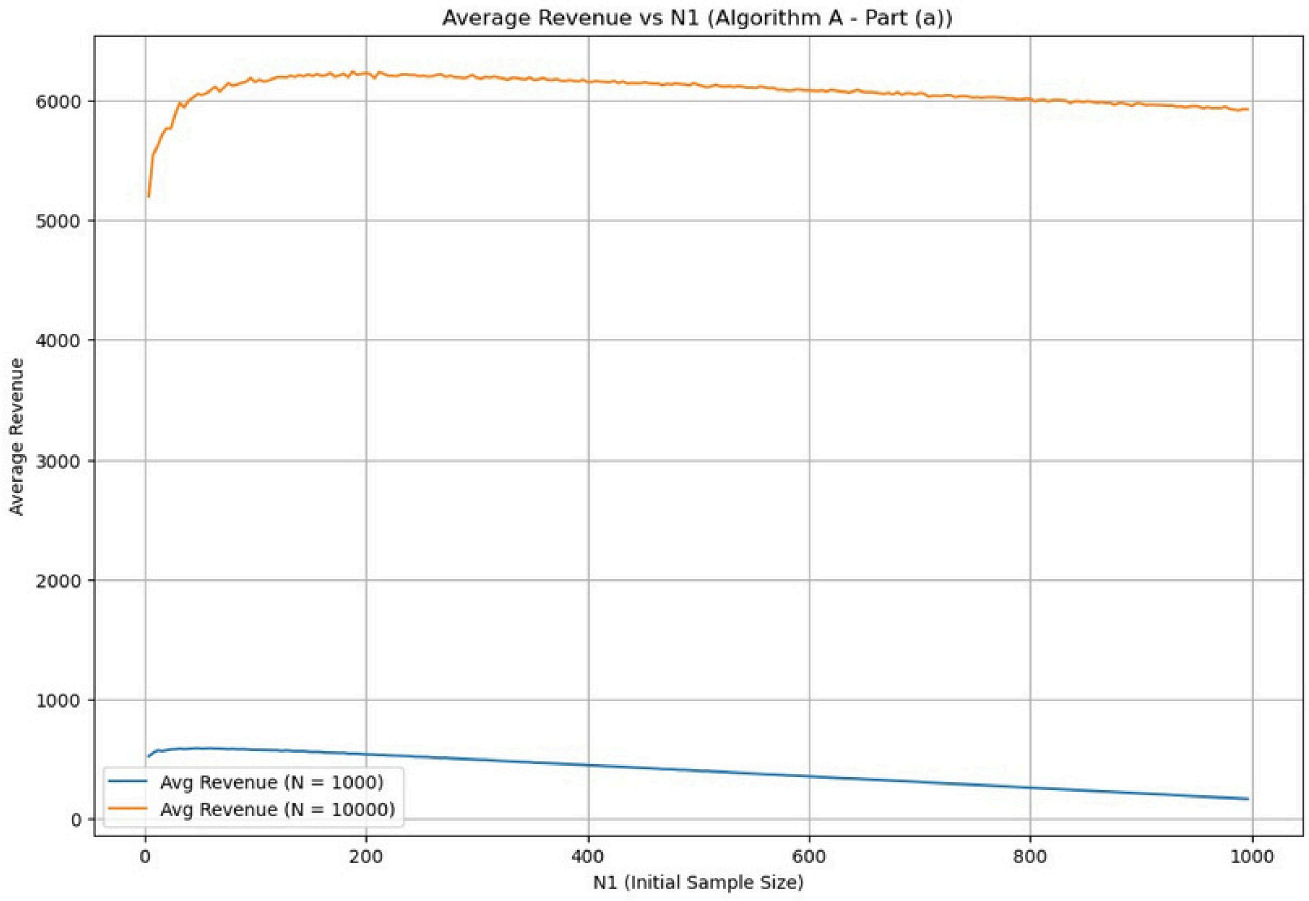


```python
import numpy as np
import matplotlib.pyplot as plt

# Parameters for System 1
K = 4
p = [0.2, 0.4, 0.6, 0.65]  # Probabilities of click for each type
a = [2, 2, 2, 2]            # Maximum revenue for each type
N_values = [1000, 10000]     # Number of users

# Function to generate random revenue for a given class k
def generate_revenue(k, size=1):
    return np.random.uniform(0, a[k], size)

# Function to simulate Algorithm A - Part (a) for different N1 values
def simulate_algorithm_A_part_a(N, N1_values, num_simulations=1000):
    avg_revenues = []

    for N1 in N1_values:
        revenues = []

        if N1 >= N:  # Ensure N1 is less than N
            continue

        for _ in range(num_simulations):
            # Initial phase: recommend randomly and calculate revenue
            R_k = np.zeros(K)
            for k in range(K):
                clicks = np.random.binomial(N1 // K, p[k])
                R_k[k] = np.sum(generate_revenue(k, clicks))

            # Exploit phase: use the best class based on initial phase
            selected_class = np.argmax(R_k)
            clicks = np.random.binomial(N - N1, p[selected_class])
            revenue = R_k[selected_class] + np.sum(generate_revenue(selected_class, clicks))

            revenues.append(revenue)
```



```python
        avg_revenues.append(np.mean(revenues))

    return avg_revenues

# Run simulations for both N values
N1_values = range(4, min(N_values), 4)  # Ensure N1 < N
plt.figure(figsize=(12, 8))

for N in N_values:
    avg_revenues = simulate_algorithm_A_part_a(N, N1_values)

    # Plot the average revenue vs N1
    plt.plot(N1_values, avg_revenues, label=f'Avg Revenue (N = {N})')

plt.title('Average Revenue vs N1 (Algorithm A - Part (a))')
plt.xlabel('N1 (Initial Sample Size)')
plt.ylabel('Average Revenue')
plt.legend()
plt.grid(True)
plt.show()
```

Plot generated:

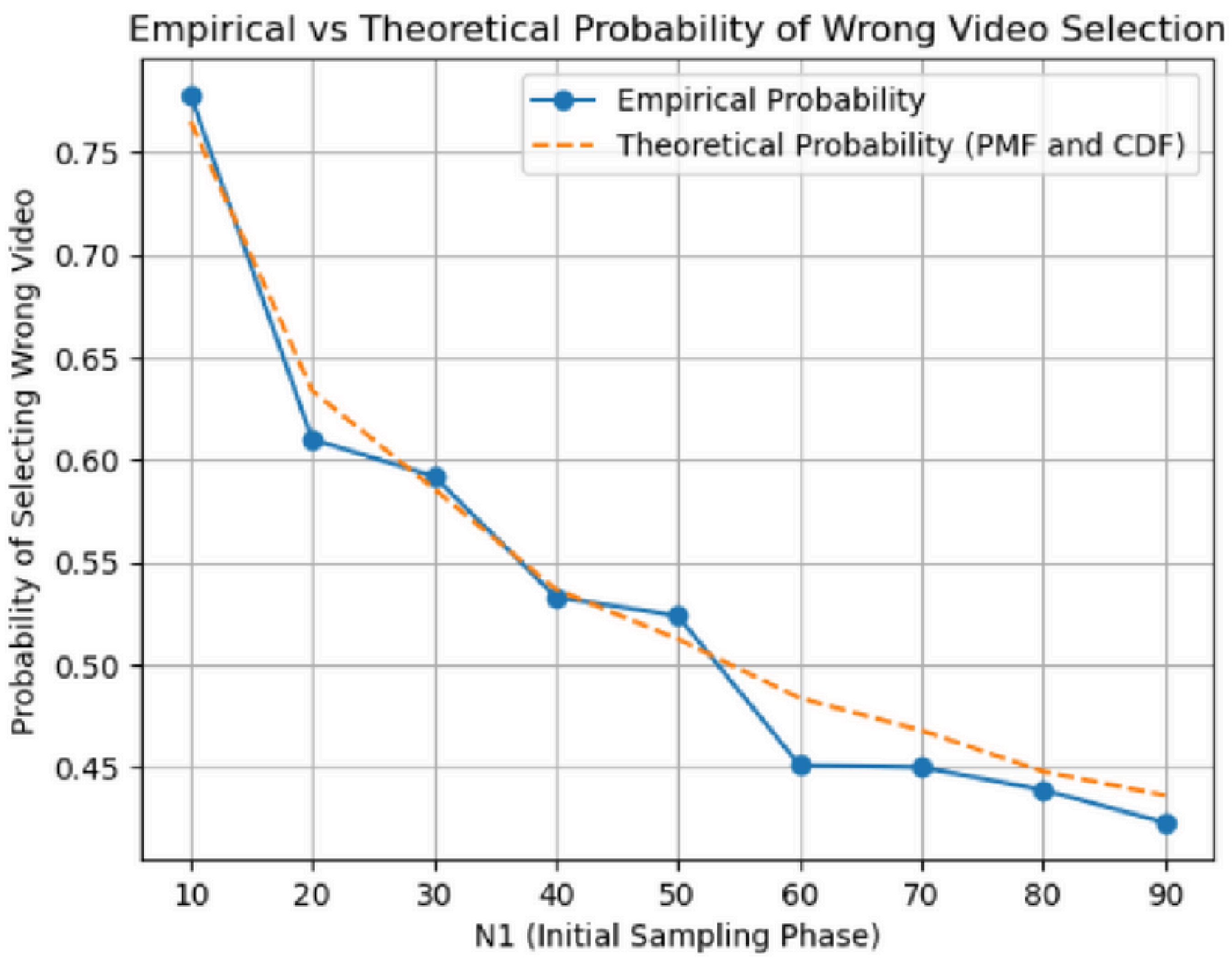


```
For N = 1000, the optimal N1 is 48 with maximum average revenue of 588.63
For N = 10000, the optimal N1 is 164 with maximum average revenue of 6237.55
```

Theoretical and empirical probabilities:

```python
import numpy as np
from scipy.stats import binom
import matplotlib.pyplot as plt

# Parameters
K = 4  # Number of videos
p = [0.2, 0.4, 0.6, 0.65]  # Click probabilities for each video
N1_values = np.arange(10, 100, 10)  # Range of N1 values
total_trials = 1000  # Number of trials to estimate empirical probability

# Theoretical probability function for wrong video selection
def theoretical_probability_of_wrong(N1):
    Pr = 0
    for i in range(1, int(N1 / 4) + 1):
        Pr += binom.pmf(i, int(N1 / 4), p[3]) * \
              binom.cdf(i - 1, int(N1 / 4), p[2]) * \
              binom.cdf(i - 1, int(N1 / 4), p[1]) * \
              binom.cdf(i - 1, int(N1 / 4), p[0])
    return 1 - Pr

# Empirical simulation for comparison
def simulate_empirical_probability(N, K, N1, p):
    wrong_selections = 0
    for _ in range(N):
        # Generate click counts for each video
        clicks = [np.random.binomial(N1 // K, p[k]) for k in range(K)]
        # Select the video with the highest clicks (correct video should be Video 4)
        best_video = np.argmax(clicks)
        if best_video != 3:
            wrong_selections += 1
    # Return the empirical probability of selecting the wrong video
    return wrong_selections / N

# Lists to store probabilities
empirical_probs = []
theoretical_probs = []
```

```python
# Compute probabilities for each N1
for N1 in N1_values:
    empirical_prob = simulate_empirical_probability(total_trials, K, N1, p)
    theoretical_prob = theoretical_probability_of_wrong(N1)
    empirical_probs.append(empirical_prob)
    theoretical_probs.append(theoretical_prob)

# Plotting the results
plt.plot(N1_values, empirical_probs, label='Empirical Probability', marker='o')
plt.plot(N1_values, theoretical_probs, label='Theoretical Probability (PMF and CDF)', linestyle='--')
plt.xlabel('N1 (Initial Sampling Phase)')
plt.ylabel('Probability of Selecting Wrong Video')
plt.title('Empirical vs Theoretical Probability of Wrong Video Selection')
plt.legend()
plt.grid(True)
plt.show()
```



Explanation:

1. Theoretical Probability:
   - We are summing over iii from 1 to N1/4 for the video with the highest probability (Video 4).
   - For each iii, we calculate the PMF of Video 4 and the CDF of Videos 3, 2, and 1.
   - The final probability is 1−sum of PMF & CDF terms

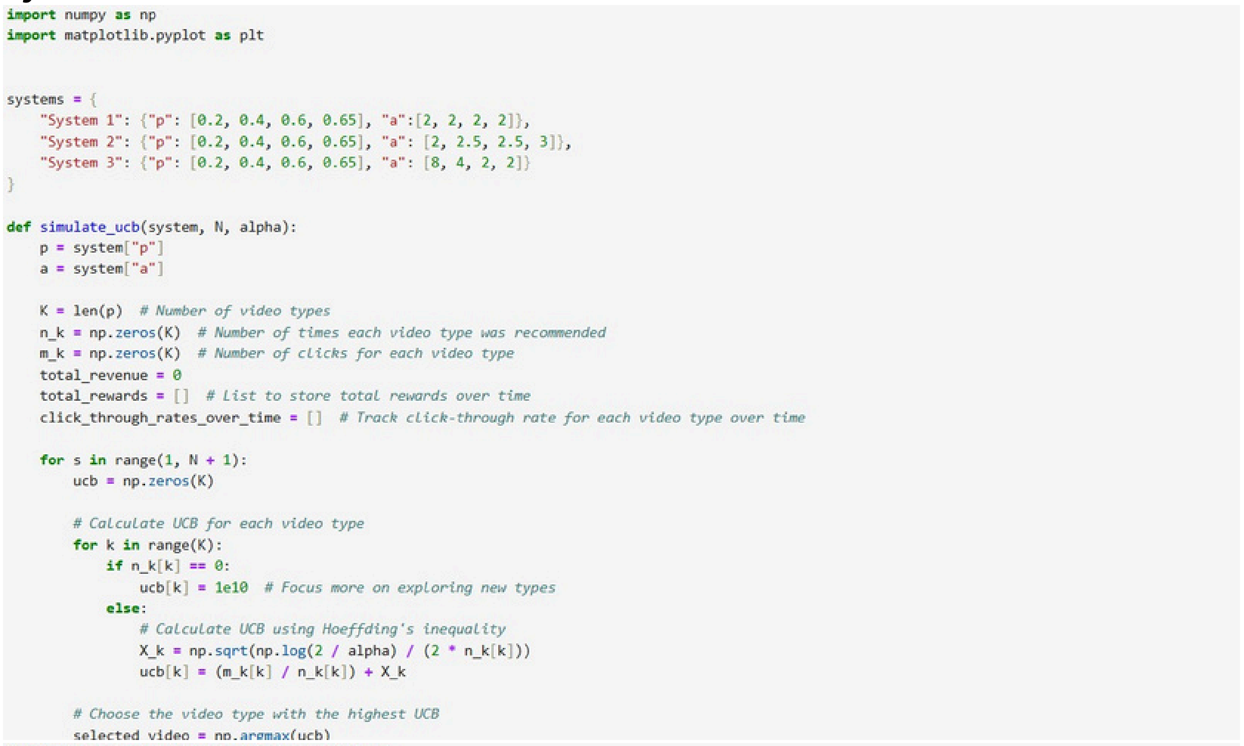
2. Empirical Probability:
   - This is computed by simulating selecting a video based on random binomial clicks. We repeat this N=1000 times and calculate the probability of choosing the wrong video.

Part 2 / Algorithm B - Problem Definition: The goal of Algorithm B is to dynamically learn the best video recommendation strategy when the click probabilities pk and revenue bounds ak for each video type are unknown. The system tries to maximize the total revenue across N users, but since it doesn't know pk (the probability of a user clicking on a video of type k) or ak (the maximum revenue possible from a video of type k), the system must learn these values over time. To achieve this, Algorithm B uses Hoe ding’s Inequality to calculate the upper confidence bound (UCB) for each pk after recommending a video. The system adaptively selects the video type with the highest UCB at each step to maximize expected revenue, refining its recommendations as more data is collected.

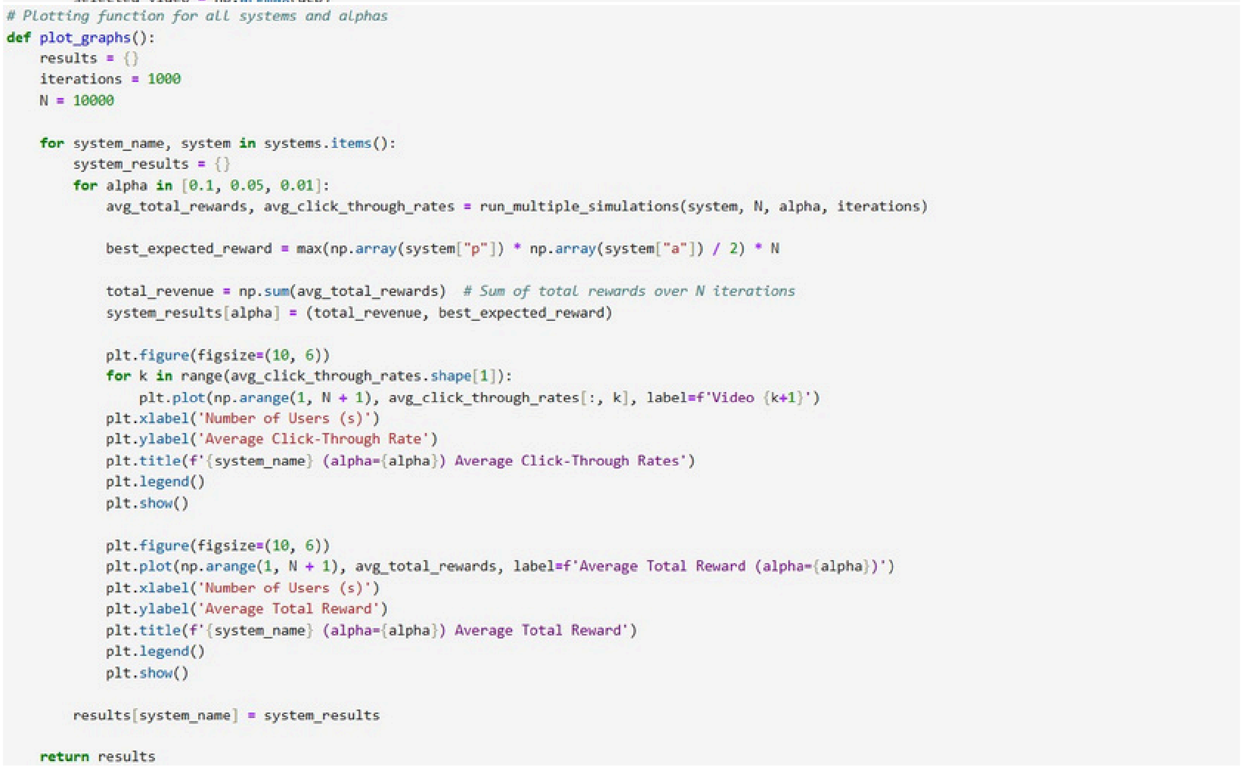
Approach for Algorithm B:

- **Initial Setup**:
  - For each video type k, track:
    - nk(s): The number of times a video of type k has been recommended after s users.
    - mk(s): The number of times a video of type k has been clicked.
    - Rk(s): The total revenue from videos of type k.
- **Hoe ding’s Inequality**:
  - At each step, calculate the upper confidence bound (UCB) for the click probability pk using Hoe ding’s Inequality:
  - The system then recommends the video type k with the highest UCB. If there is a tie, it breaks it randomly
- **Adaptive Learning**:
  - After each recommendation, the system updates the values of mk(s), nk(s), and Rk(s) and repeats the process for the next user.
  - Over time, the system converges towards recommending the video type with the highest expected revenue, while balancing exploration (trying different video types to refine estimates) and exploitation (recommending the video type that seems most profitable).

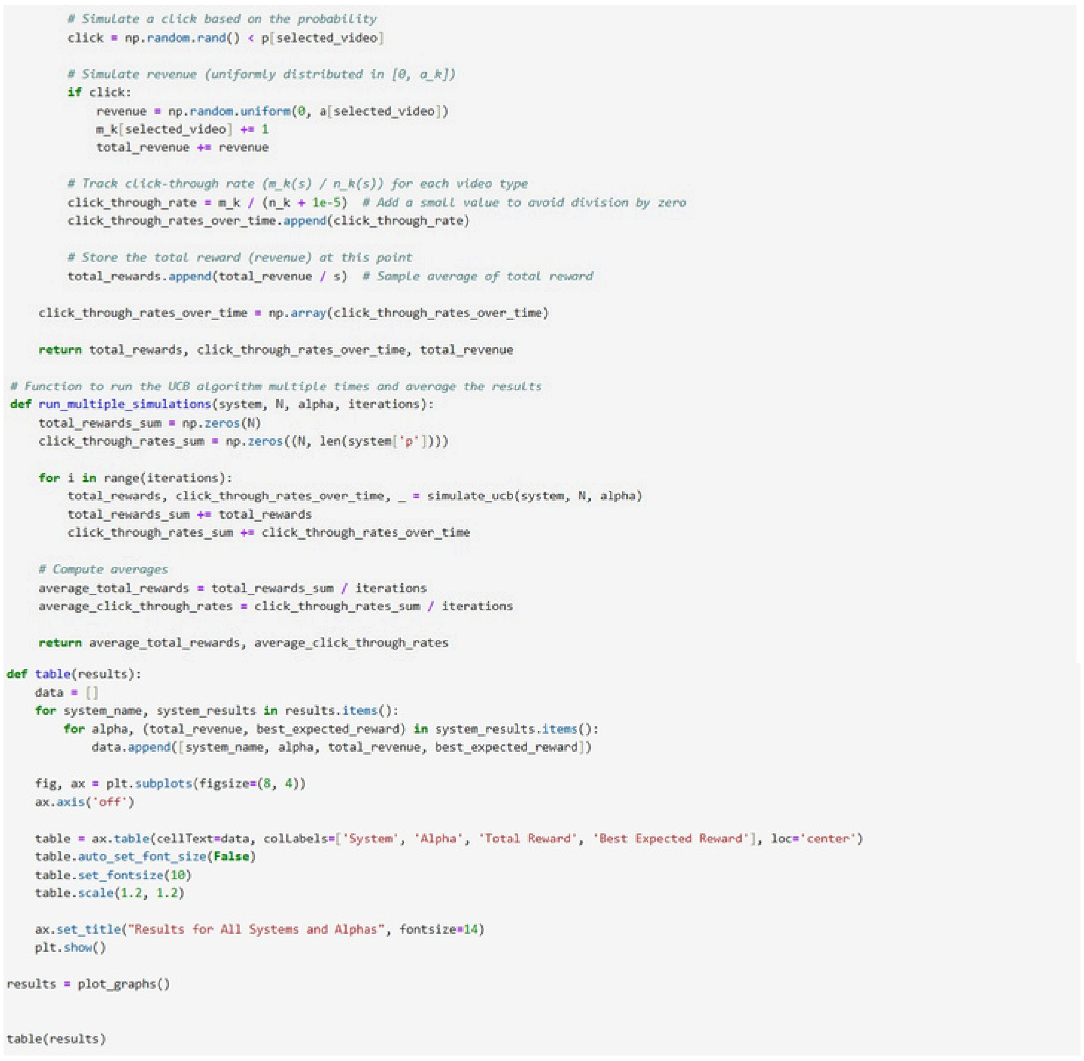
- **Python Code:**



```python
import numpy as np
import matplotlib.pyplot as plt


systems = {
    "System 1": {"p": [0.2, 0.4, 0.6, 0.65], "a":[2, 2, 2, 2]},
    "System 2": {"p": [0.2, 0.4, 0.6, 0.65], "a": [2, 2.5, 2.5, 3]},
    "System 3": {"p": [0.2, 0.4, 0.6, 0.65], "a": [8, 4, 2, 2]}
}

def simulate_ucb(system, N, alpha):
    p = system["p"]
    a = system["a"]

    K = len(p)  # Number of video types
    n_k = np.zeros(K)  # Number of times each video type was recommended
    m_k = np.zeros(K)  # Number of clicks for each video type
    total_revenue = 0
    total_rewards = []  # List to store total rewards over time
    click_through_rates_over_time = []  # Track click-through rate for each video type over time

    for s in range(1, N + 1):
        ucb = np.zeros(K)

        # Calculate UCB for each video type
        for k in range(K):
            if n_k[k] == 0:
                ucb[k] = 1e10  # Focus more on exploring new types
            else:
                # Calculate UCB using Hoeffding's inequality
                X_k = np.sqrt(np.log(2 / alpha) / (2 * n_k[k]))
                ucb[k] = (m_k[k] / n_k[k]) + X_k

        # Choose the video type with the highest UCB
        selected_video = np.argmax(ucb)
```

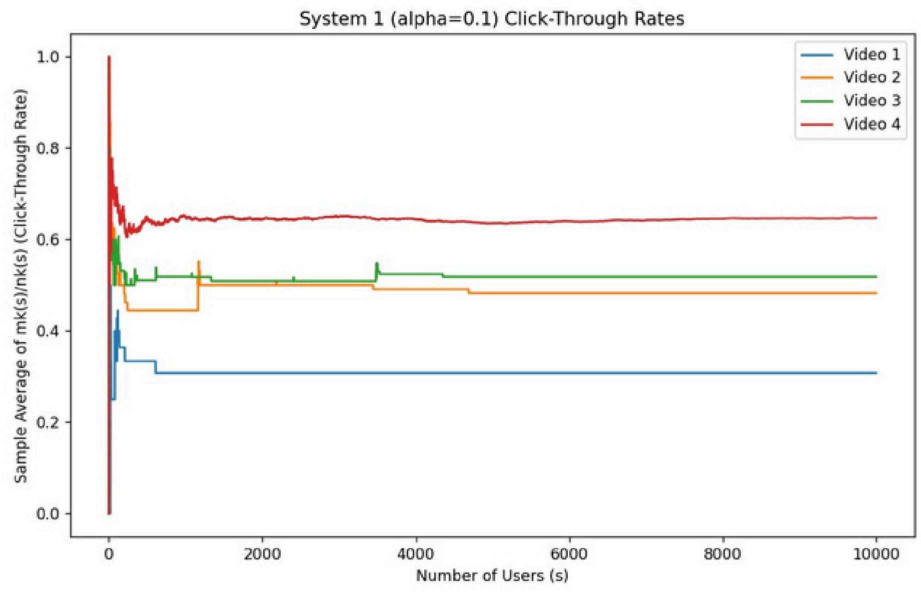


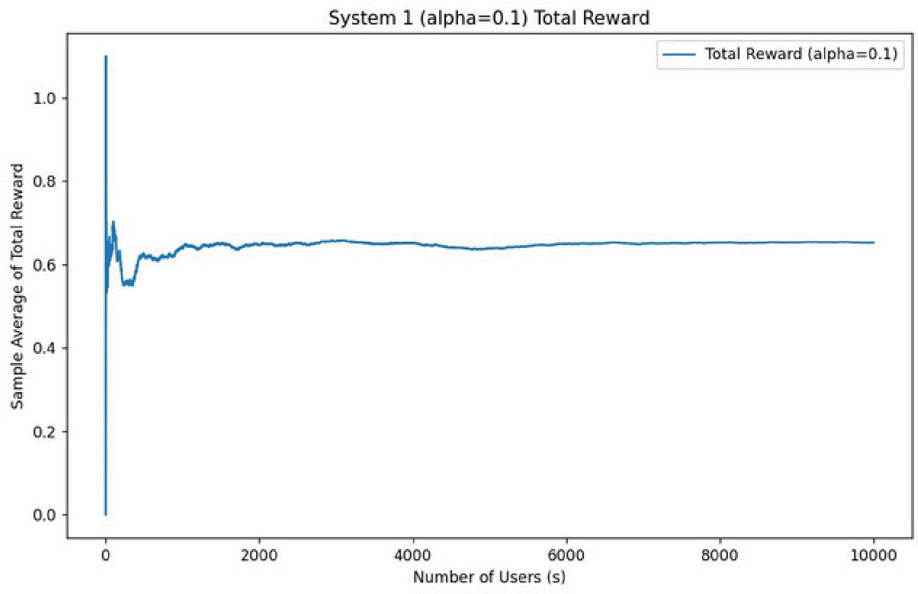
```python
# Plotting function for all systems and alphas
def plot_graphs():
    results = {}
    iterations = 1000
    N = 10000

    for system_name, system in systems.items():
        system_results = {}
        for alpha in [0.1, 0.05, 0.01]:
            avg_total_rewards, avg_click_through_rates = run_multiple_simulations(system, N, alpha, iterations)

            best_expected_reward = max(np.array(system["p"]) * np.array(system["a"]) / 2) * N

            total_revenue = np.sum(avg_total_rewards)  # Sum of total rewards over N iterations
            system_results[alpha] = (total_revenue, best_expected_reward)

            plt.figure(figsize=(10, 6))
            for k in range(avg_click_through_rates.shape[1]):
                plt.plot(np.arange(1, N + 1), avg_click_through_rates[:, k], label=f'Video {k+1}')
            plt.xlabel('Number of Users (s)')
            plt.ylabel('Average Click-Through Rate')
            plt.title(f'{system_name} (alpha={alpha}) Average Click-Through Rates')
            plt.legend()
            plt.show()

            plt.figure(figsize=(10, 6))
            plt.plot(np.arange(1, N + 1), avg_total_rewards, label=f'Average Total Reward (alpha={alpha})')
            plt.xlabel('Number of Users (s)')
            plt.ylabel('Average Total Reward')
            plt.title(f'{system_name} (alpha={alpha}) Average Total Reward')
            plt.legend()
            plt.show()

        results[system_name] = system_results

    return results
```



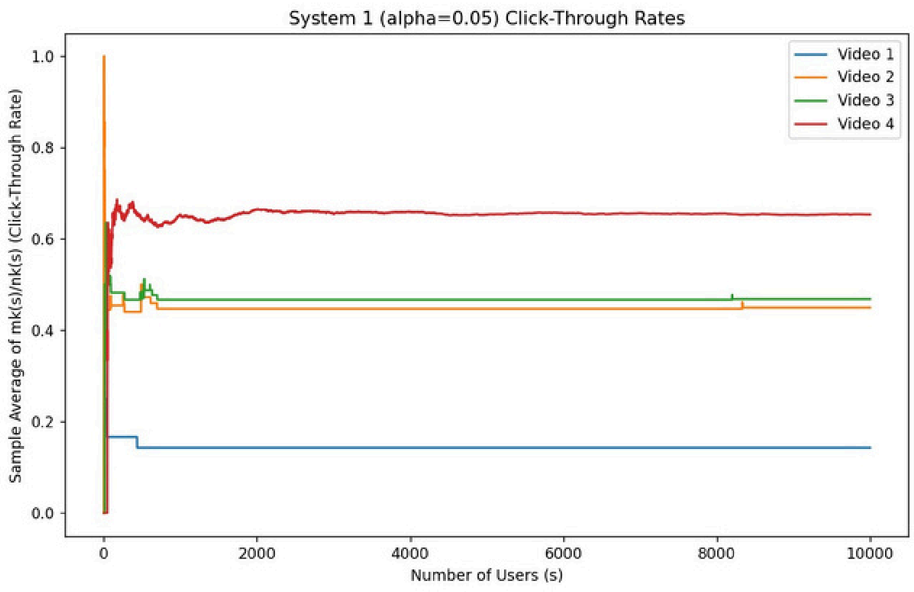
```python
        # Simulate a click based on the probability
        click = np.random.rand() < p[selected_video]

        # Simulate revenue (uniformly distributed in [0, a_k])
        if click:
            revenue = np.random.uniform(0, a[selected_video])
            m_k[selected_video] += 1
            total_revenue += revenue

        # Track click-through rate (m_k(s) / n_k(s)) for each video type
        click_through_rate = m_k / (n_k + 1e-5)  # Add a small value to avoid division by zero
        click_through_rates_over_time.append(click_through_rate)

        # Store the total reward (revenue) at this point
        total_rewards.append(total_revenue / s)  # Sample average of total reward

    click_through_rates_over_time = np.array(click_through_rates_over_time)

    return total_rewards, click_through_rates_over_time, total_revenue

# Function to run the UCB algorithm multiple times and average the results
def run_multiple_simulations(system, N, alpha, iterations):
    total_rewards_sum = np.zeros(N)
    click_through_rates_sum = np.zeros((N, len(system['p'])))

    for i in range(iterations):
        total_rewards, click_through_rates_over_time, _ = simulate_ucb(system, N, alpha)
        total_rewards_sum += total_rewards
        click_through_rates_sum += click_through_rates_over_time

    # Compute averages
    average_total_rewards = total_rewards_sum / iterations
    average_click_through_rates = click_through_rates_sum / iterations

    return average_total_rewards, average_click_through_rates

def table(results):
    data = []
    for system_name, system_results in results.items():
        for alpha, (total_revenue, best_expected_reward) in system_results.items():
            data.append([system_name, alpha, total_revenue, best_expected_reward])

    fig, ax = plt.subplots(figsize=(8, 4))
    ax.axis('off')

    table = ax.table(cellText=data, colLabels=['System', 'Alpha', 'Total Reward', 'Best Expected Reward'], loc='center')
    table.auto_set_font_size(False)
    table.set_fontsize(10)
    table.scale(1.2, 1.2)

    ax.set_title("Results for All Systems and Alphas", fontsize=14)
    plt.show()

results = plot_graphs()


table(results)
```

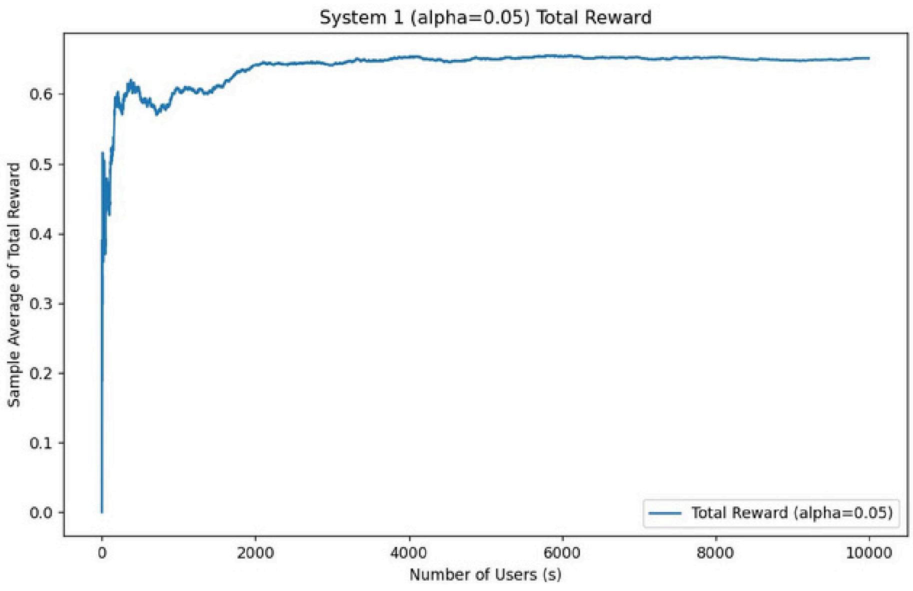
(a) α = 0.1


System 1 (alpha=0.1) Click-Through Rates
Video 1
Video 2
Video 3
Video 4
Sample Average of mk(s)/nk(s) (Click-Through Rate)
Number of Users (s)
0.0
0.2
0.4
0.6
0.8
1.0
0
2000
4000
6000
8000
10000

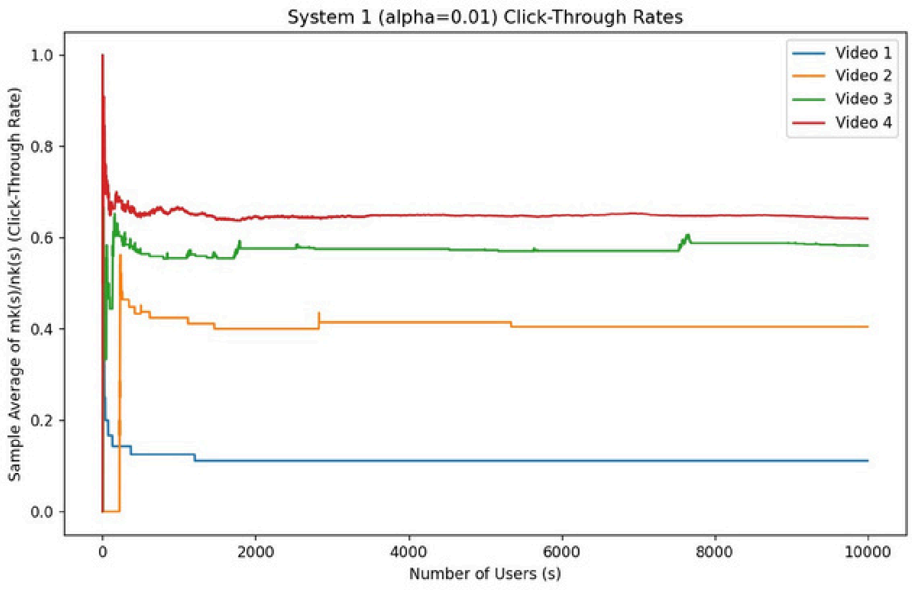

System 1 (alpha=0.1) Total Reward
Total Reward (alpha=0.1)
Sample Average of Total Reward
Number of Users (s)
0.0
0.2
0.4
0.6
0.8
1.0
0
2000
4000
6000
8000
10000

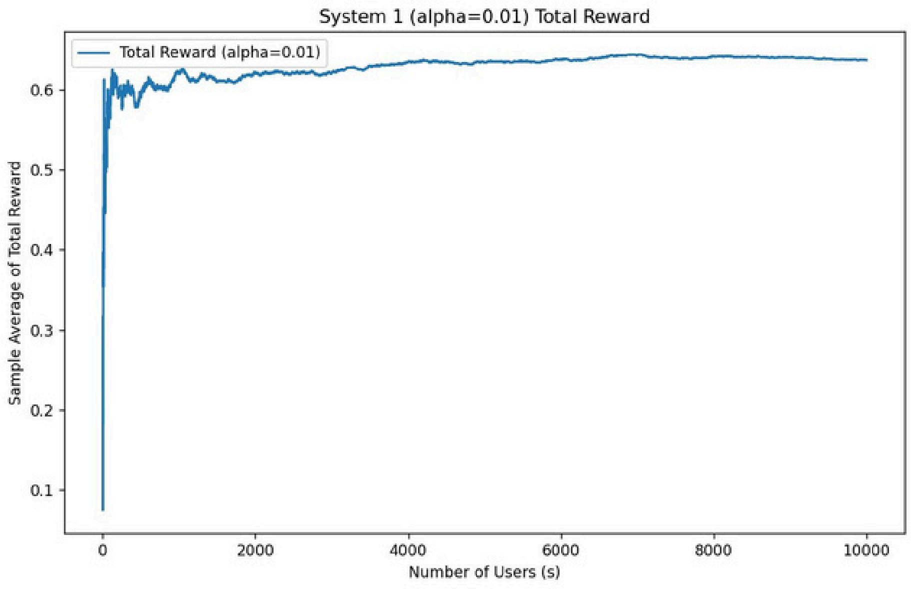
α = 0.05


System 1 (alpha=0.05) Click-Through Rates
Video 1
Video 2
Video 3
Video 4
Sample Average of mk(s)/nk(s) (Click-Through Rate)
Number of Users (s)
0.0
0.2
0.4
0.6
0.8
1.0
0
2000
4000
6000
8000
10000


System 1 (alpha=0.05) Total Reward
Total Reward (alpha=0.05)
Sample Average of Total Reward
Number of Users (s)
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0
2000
4000
6000
8000
10000

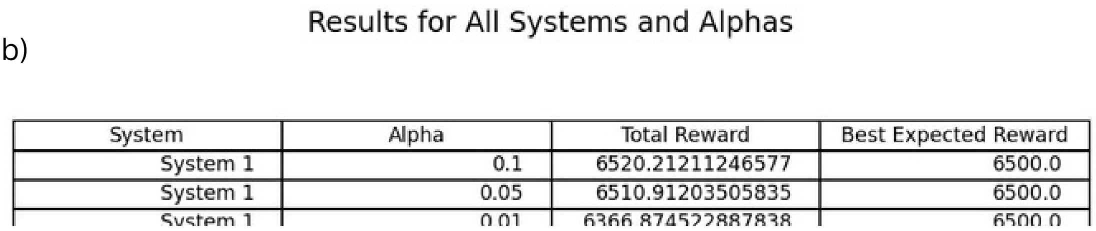
α = 0.01


System 1 (alpha=0.01) Click-Through Rates
Video 1
Video 2
Video 3
Video 4
Sample Average of mk(s)/nk(s) (Click-Through Rate)
Number of Users (s)
0.0
0.2
0.4
0.6
0.8
1.0
0
2000
4000
6000
8000
10000


System 1 (alpha=0.01) Total Reward
Total Reward (alpha=0.01)
Sample Average of Total Reward
Number of Users (s)
0.1
0.2
0.3
0.4
0.5
0.6
0
2000
4000
6000
8000
10000

(b)

(b)


Results for All Systems and Alphas

System	Alpha	Total Reward	Best Expected Reward
System 1	0.1	6520.21211246577	6500.0
System 1	0.05	6510.91203505835	6500.0
System 1	0.01	6366.874522887838	6500.0

(c)

The numerical computation of average rewards is mentioned as the total rewards in table of part b.

The parameters α, N, and p significantly impact the performance of Algorithm B.

- Effect of α: This parameter regulates the balance between exploration and exploitation. A smaller α (e.g., 0.01) encourages exploitation, where the algorithm focuses on known high-reward options. While this can speed up convergence, it risks locking into suboptimal choices too early. On the other hand, a larger α (e.g., 0.1) increases exploration, allowing the algorithm to search for better options. However, this can slow down convergence and reduce immediate rewards.

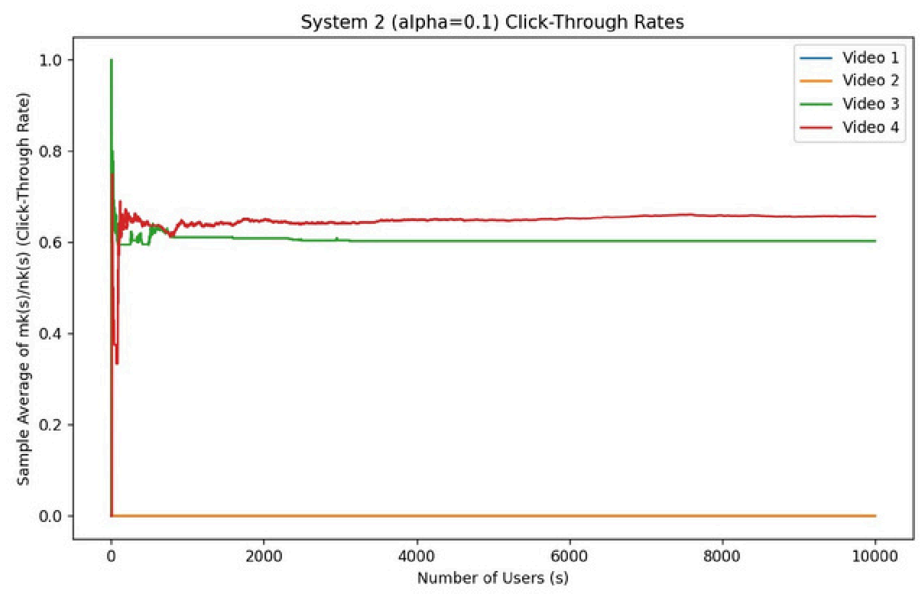
- Effect of N: As the number of users (N) increases, the algorithm gains more data to estimate reward probabilities more accurately, improving overall performance. With fewer users, the uncertainty increases, leading to a higher chance of making suboptimal choices. Our tests with varying N values align with this theoretical expectation.

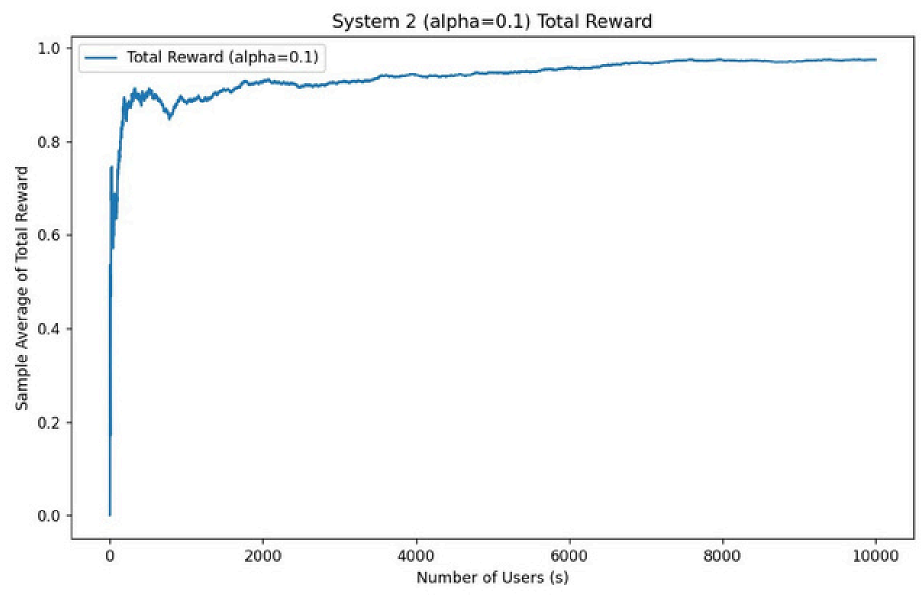
- Effect of p: The reward probabilities (p) play a key role in identifying the best options. When certain items have higher pk values, rewards are generated more frequently, and larger differences between pk values enable faster identification of the best-performing items. Conversely, when the differences between pk values are small or the probabilities are low, the exploration phase is prolonged, reducing the total reward. This effect is confirmed using System 1.

In summary, a smaller α and larger N lead to quicker convergence and higher short-term rewards, while a larger α favors more exploration for improved long-term performance. The disparity in pk values determines how efficiently the algorithm identifies the top options.

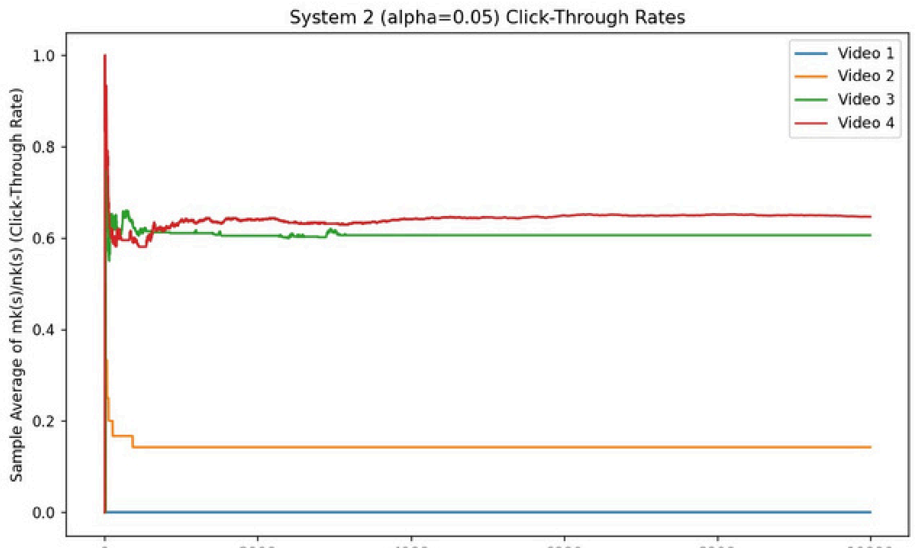
(d) System 2

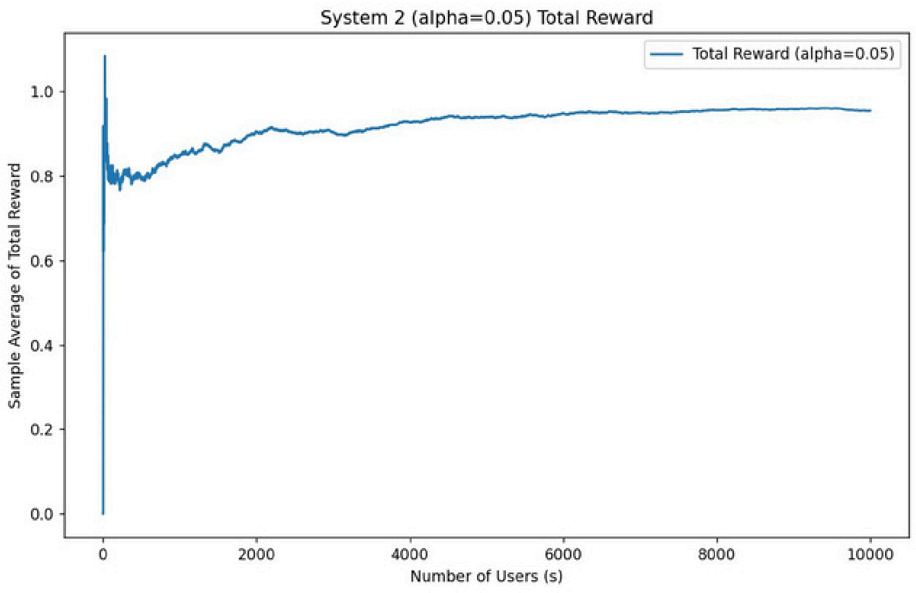
α = 0.1



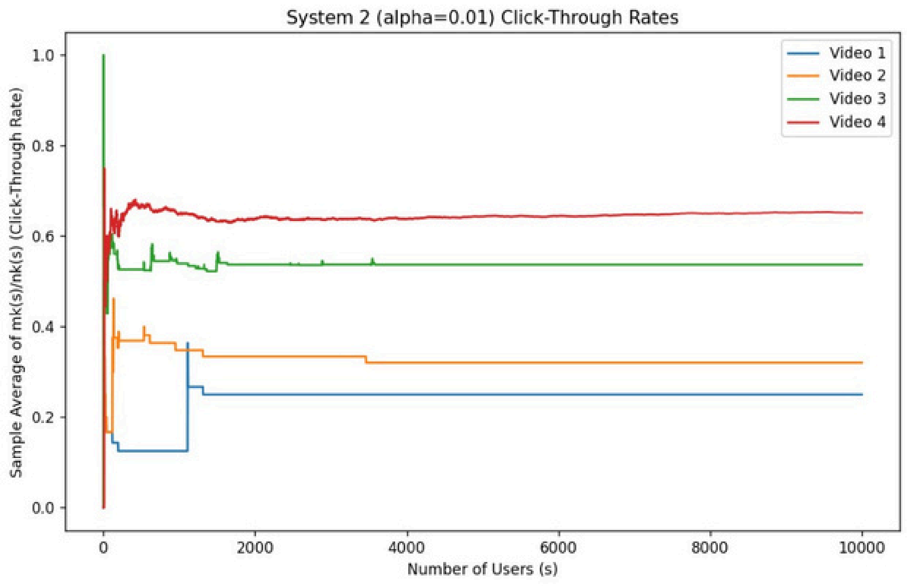


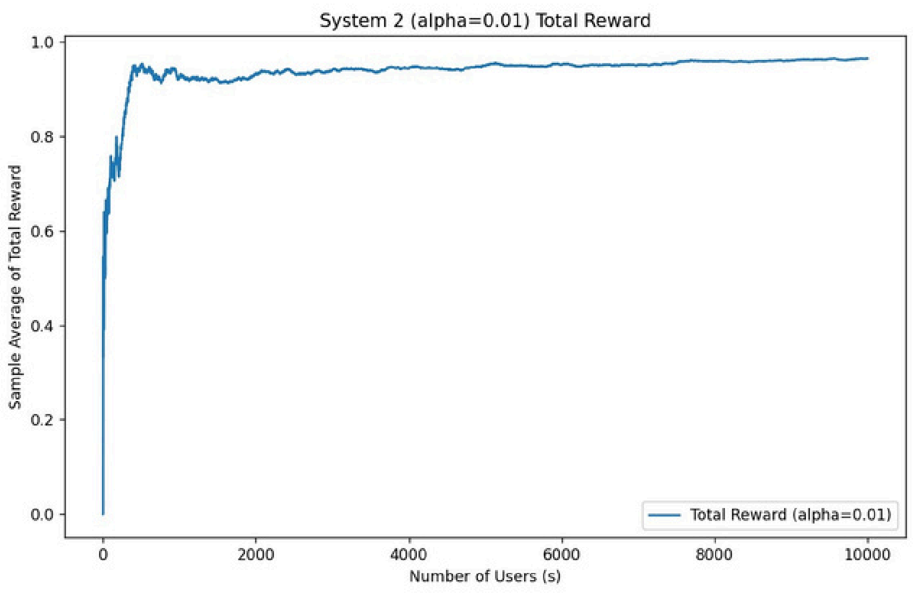
α = 0.05


System 2 (alpha=0.05) Click-Through Rates
Sample Average of mk(s)/nk(s) (Click-Through Rate)
1.0
0.8
0.6
0.4
0.2
0.0
Video 1
Video 2
Video 3
Video 4


System 2 (alpha=0.05) Total Reward
Sample Average of Total Reward
Total Reward (alpha=0.05)
1.0
0.8
0.6
0.4
0.2
0.0
0
2000
4000
6000
8000
10000
Number of Users (s)

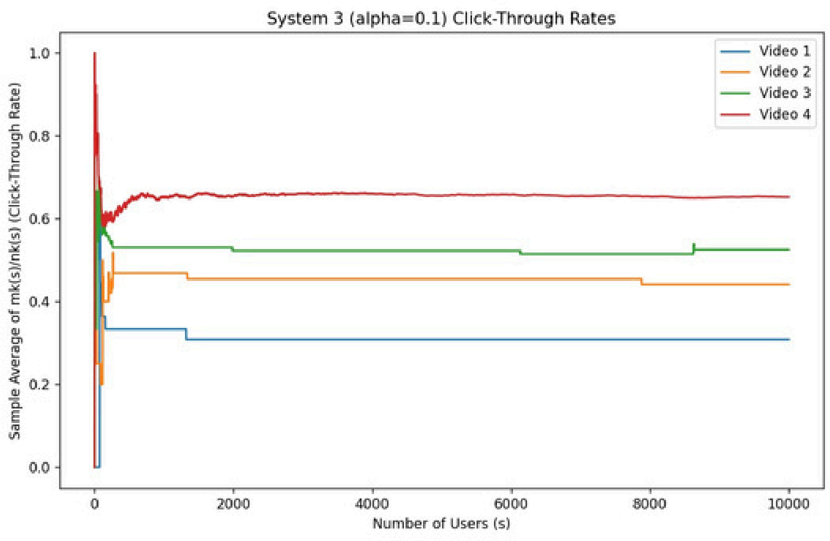
α = 0.01

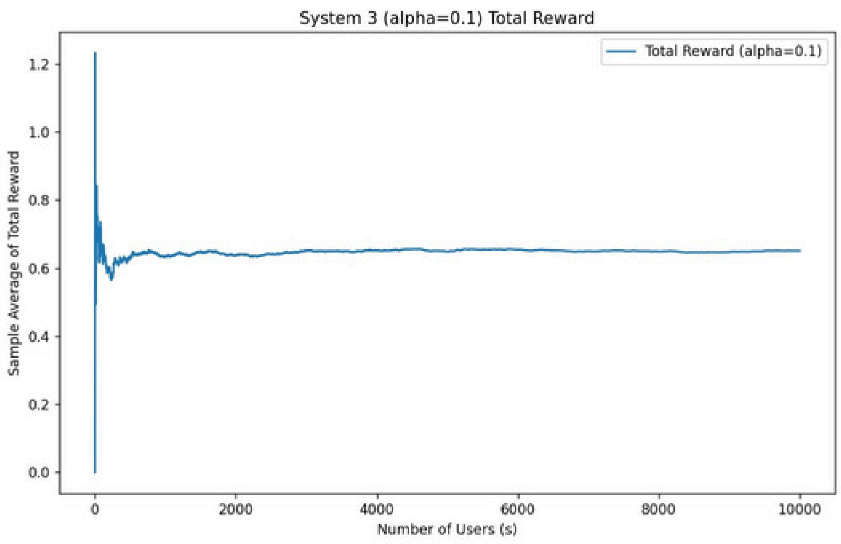

System 2 (alpha=0.01) Click-Through Rates
Sample Average of mk(s)/nk(s) (Click-Through Rate)
Video 1
Video 2
Video 3
Video 4
1.0
0.8
0.6
0.4
0.2
0.0
0
2000
4000
6000
8000
10000
Number of Users (s)


System 2 (alpha=0.01) Total Reward
Sample Average of Total Reward
Number of Users (s)
Total Reward (alpha=0.01)
0.0
0.2
0.4
0.6
0.8
1.0
0
2000
4000
6000
8000
10000

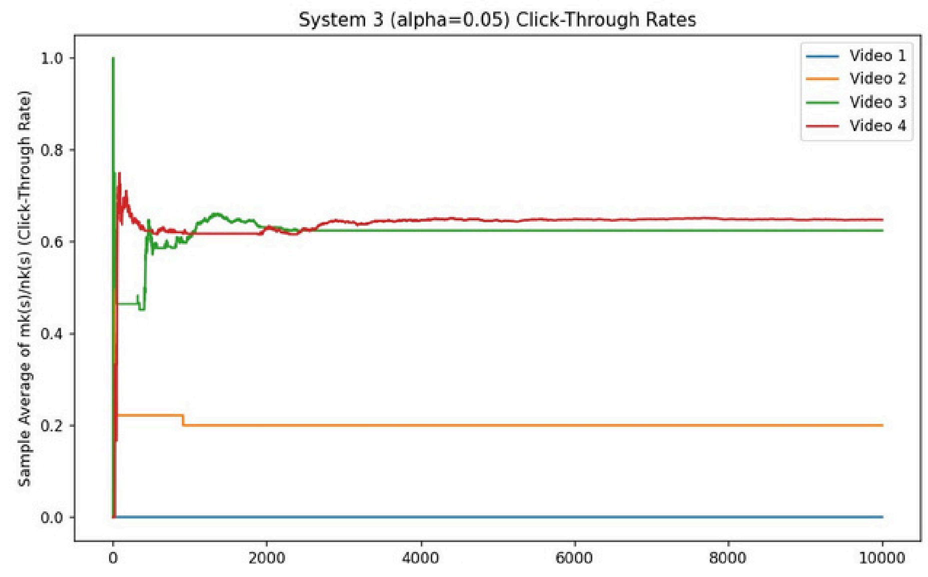
α = 0.1

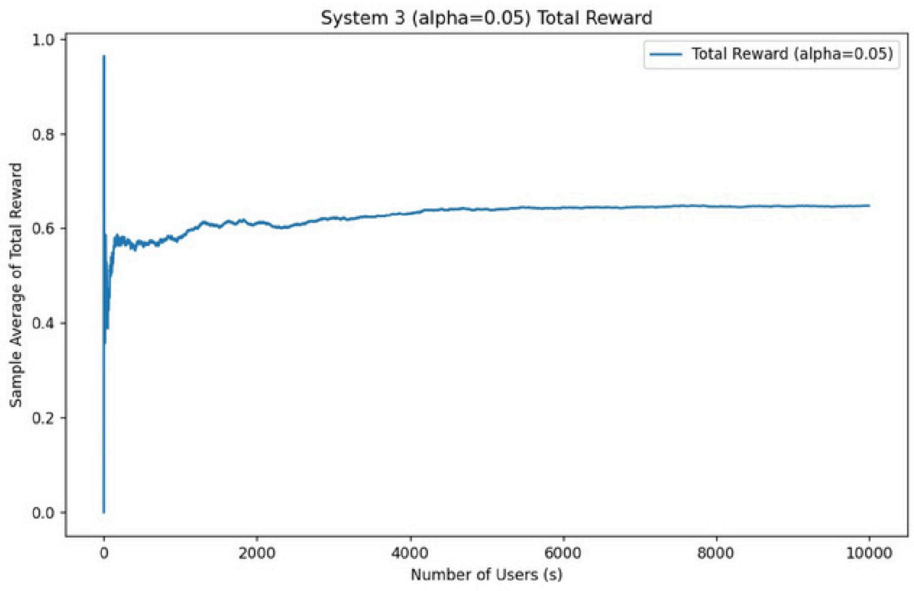
System 3


System 3 (alpha=0.1) Click-Through Rates
Sample Average of mk(s)/nk(s) (Click-Through Rate)
Number of Users (s)
Video 1
Video 2
Video 3
Video 4
0.0
0.2
0.4
0.6
0.8
1.0
0
2000
4000
6000
8000
10000

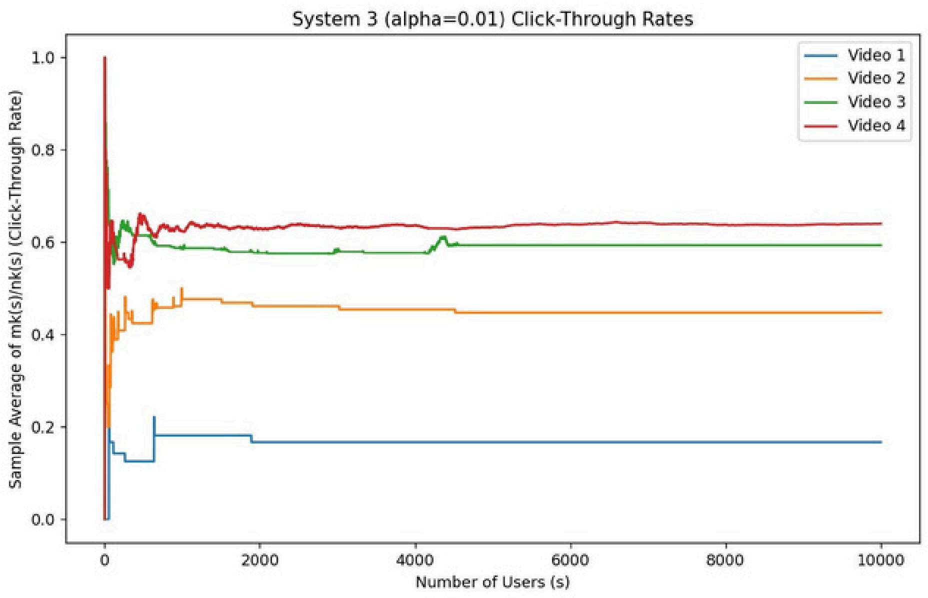

System 3 (alpha=0.1) Total Reward
Sample Average of Total Reward
Number of Users (s)
Total Reward (alpha=0.1)
0.0
0.2
0.4
0.6
0.8
1.0
1.2
0
2000
4000
6000
8000
10000

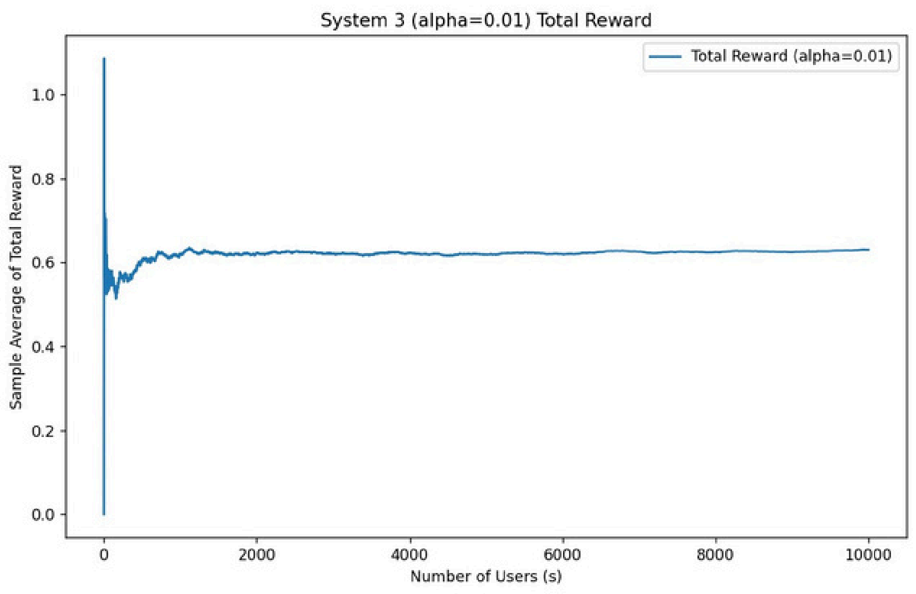
α = 0.05


System 3 (alpha=0.05) Click-Through Rates
Sample Average of mk(s)/nk(s) (Click-Through Rate)
Video 1
Video 2
Video 3
Video 4
1.0
0.8
0.6
0.4
0.2
0.0
0
2000
4000
6000
8000
10000


System 3 (alpha=0.05) Total Reward
Sample Average of Total Reward
Total Reward (alpha=0.05)
Number of Users (s)
1.0
0.8
0.6
0.4
0.2
0.0
0
2000
4000
6000
8000
10000

α = 0.01


System 3 (alpha=0.01) Click-Through Rates
Sample Average of mk(s)/nk(s) (Click-Through Rate)
Number of Users (s)
Video 1
Video 2
Video 3
Video 4
1.0
0.8
0.6
0.4
0.2
0.0
0
2000
4000
6000
8000
10000



(e)

- Assuming that user preferences stay the same as they continue using a system is a bit too simple. In reality, people's tastes can change over time for many reasons. They might get bored, influenced by new trends, or pick up on external factors that shape their choices. As users interact with recommendation systems, they often learn more about what they like and refine their preferences, making the whole process dynamic and ever-evolving. So, this assumption doesn't fully capture the complexity of how people really behave.

- Instead of assuming static Pk, model Pk as a time-varying function. Over time, users may grow more or less interested in certain types of videos based on their recommendation history.Linear Decay/Growth: Assume that exposure to the same type of video decreases (or increases) the likelihood of clicking on it over time.

$$p_k(s) = p_k(0) - \beta \cdot s$$

β is a decay factor
s is the number of times type k has been recommended

- We can model a recommendation system with a memory factor that adjusts the probability Pk of recommending videos based on recent user interactions. If a user clicks on a video of type k, Pk increases, prioritizing similar videos. However, repeated recommendations without clicks cause Pk to decrease, reflecting reduced interest. This memory factor dynamically decays over time, ensuring the system adapts to both recent and long-term user behavior.