Multi-Armed Bandit with Costs

Imagine you are managing a fleet of delivery drones, each assigned to one of several routes (arms). Each route provides a reward based on its efficiency (e.g., time saved or customer satisfaction), but it also incurs a cost (e.g., fuel consumption). Your task is to design an algorithm that selects routes to maximize the net reward, defined as: Net Reward = Reward – Cost

You will simulate the multi-armed bandit problem with costs using Python and analyze the performance of your algorithm.

Exercise Details

1. Problem Setup

- There are k = 5 routes (arms).
- Each route i has:
  - A reward distribution: $R_i = N\{mu_i, sigma_i^2)$
  - A fixed cost: C_i , representing the cost of selecting the route.
  - The net reward for selecting arm i is: Net $Reward_i = R_i - C_i$

2. Parameters

- Mean rewards mu: [0.8, 0.6, 0.9, 0.4, 0.7]
- Standard deviations sigma: [0.1, 0.1, 0.1, 0.1, 0.1]
- Costs C : [0.2, 0.1, 0.3, 0.05, 0.15]

3. Goal: Maximize the cumulative net reward over T = 1000 steps by selecting routes (arms) using an appropriate multi-armed bandit algorithm.

Task

Step 1: Implement the Environment

Write a function to simulate the environment:

```
import numpy as np

def get_reward_and_cost(arm):
    """
    Simulates the environment by returning the reward and cost for a given arm.

    Parameters:
        arm (int): The selected arm (route).

    Returns:
        reward (float): Sampled reward from the arm's reward distribution.
        cost (float): Fixed cost of the arm.
    """
    # Define parameters for each arm
    reward_means = [0.8, 0.6, 0.9, 0.4, 0.7]
    reward_stds = [0.1, 0.1, 0.1, 0.1, 0.1]
    costs = [0.2, 0.1, 0.3, 0.05, 0.15]

    # Sample reward from the normal distribution
    reward = np.random.normal(reward_means[arm], reward_stds[arm])

    # Return reward and cost
    return reward, costs[arm]
```

Step 2: Implement the ε-Greedy Algorithm. Use the net reward (R_i - C_i) to update the Q-values.

Step 3: Run the Simulation
Run the ε-greedy algorithm and visualize the results:

```
import matplotlib.pyplot as plt

# Parameters
num_arms = 5
num_steps = 1000
epsilon = 0.1

# Run the algorithm
cumulative_net_rewards = epsilon_greedy_with_costs(num_arms, num_steps, epsilon)

# Plot the results
plt.plot(cumulative_net_rewards)
plt.xlabel("Steps")
plt.ylabel("Cumulative Net Reward")
plt.title("ε-Greedy Algorithm with Costs")
plt.show()
```

Answer the questions:

1. Algorithm Design:
   - Explain why we use net rewards (R_i - C_i) instead of raw rewards (R_i) in this problem.
   - How does the ε-greedy algorithm balance exploration and exploitation? What happens if epsilon is too high or too low?

2. Experimentation:
   - Run the simulation with different values of epsilon (e.g., 0.01, 0.1, 0.5). How does the choice of epsilon affect the cumulative net reward?
   - Modify the costs (C_i) to make one arm significantly more expensive. How does this impact the algorithm's performance?

3. Comparison:
- Non-Stationary Costs: Modify the environment so that the costs (C_i) change over time. For example, let C_i drift randomly every 100 steps. Update the algorithm to handle this non-stationarity.
- Implement UCB algorithm and compare its performance with ε-greedy. Which algorithm performs better in terms of cumulative net reward?
- Dynamic Rewards: Introduce changes in the reward distributions (R_i) over time. For example, let the mean rewards (mu_i) shift periodically. Analyze how the algorithm adapts to these changes.

4. Analysis:
   - Calculate the regret for the ε-greedy algorithm. Regret is defined as the difference between the cumulative net reward of the optimal arm and the cumulative net reward obtained by the algorithm.
   - Discuss how the regret grows over time. Is the growth linear or sublinear?