In this section, the paper introduces the **Combinatorial Multi-armed Bandit with Probabilistically Triggered Arms (CMAB-T)**, which expands upon the standard CMAB problem by incorporating probabilistic feedback where not all base arms in a chosen combinatorial action may be triggered. The problem setup involves a set of base arms, a combinatorial action space, and a reward function where the feedback from each base arm is probabilistically triggered depending on the action chosen. This setup is particularly relevant for real-world applications like recommendation systems and resource allocation, where the probability of observing the outcome of each action varies.

The key challenge of CMAB-T lies in the combinatorial structure of actions and the probabilistic nature of feedback, where each selected action does not guarantee full observation of all involved base arms. The paper defines a reward function that takes into account the triggered arms and their outcomes. The combination of combinatorial decision-making and probabilistic triggering introduces unique complexities, including the need to estimate expected rewards with partial observations. The paper aims to minimize the **suboptimality gap**, which is the difference between the expected reward of the chosen action and the optimal action, by leveraging a novel algorithm that addresses the uncertainties arising from the probabilistic feedback.

The section further outlines **two critical conditions** for ensuring efficient learning: **monotonicity** (reward increases with the mean of the base arms) and **bounded smoothness** (the sensitivity of rewards to changes in the base arms' outcomes is controlled by the triggering probabilities). These conditions enable the algorithm to make robust decisions despite limited and incomplete feedback. Additionally, the paper introduces a theoretical framework to quantify the amount of data required for accurate reward estimation, and the proposed approach is proven to achieve near-optimal performance under these conditions.

Related Formulas:

1. **Suboptimality Gap** The suboptimality gap measures how close the chosen action is to the optimal action. It is defined as the difference in expected rewards:

$$\text{SubOpt}(S^{\hat{}}; \alpha, I) = \alpha \cdot r(S^{*}; \mu) - r(S^{\hat{}}; \mu)$$

   where S^{*} is the optimal action and μ is the mean vector of the base arms.

2. **Monotonicity Condition** The reward is monotonically increasing with respect to the mean vector μ of the base arms:

$$\text{If } \mu_i \leq \mu'_i \text{ for all } i, r(S; \mu) \leq r(S; \mu')$$

3. **1-norm TPM Bounded Smoothness** The reward function's sensitivity to changes in the base arm means is controlled by the smoothness coefficient B_1:

$$\left| r(S; \mu') - r(S; \mu) \right) \leq B_1 \sum_{i \in [m]} p_{D_{\text{arm}}, S_i} \left| \mu_i - \mu'_i \right)$$

   where p_{D_{arm}, S_i} is the probability of arm i being triggered for action S.

Pseudocode for the **CLCB Algorithm**:

```
def CLCB(D, oracle, delta):
    # Input: Dataset D, oracle for combinatorial actions, failure
probability delta
    for arm in range(m):  # Iterate over each base arm
        # Step 1: Compute the number of times each arm is triggered
        Ni = sum([1 for t in range(n) if i in tau_t])
        # Step 2: Compute empirical mean for each arm
        mu_hat_i = sum([Xt_i for t in range(n) if i in tau_t]) / Ni
        # Step 3: Compute the lower confidence bound for each arm
        LCB_mu_i = mu_hat_i - sqrt(log(4 * m * n / delta) / (2 * Ni))

    # Step 4: Call oracle to select the combinatorial action
    S_hat = oracle(LCB_mu_1, ..., LCB_mu_m)

    return S_hat  # Return the selected action
```

To simulate `CLCB` function, let's assume a simplified example with synthetic data for the dataset `D` and a basic oracle function. The oracle will simply return the action (super arm) corresponding to the arm with the highest lower confidence bound (LCB).

Here's how you can define and run the `CLCB` function:

Steps:

1. **Dataset `D`**: We'll generate a dataset where each sample consists of a list of outcomes for the base arms and a set of triggered arms (i.e., which arms have feedback for that particular sample).
2. **Oracle function**: The oracle will select the combinatorial action that maximizes the lower confidence bound from the list of LCBs.
3. **Run the `CLCB` function**: We'll pass the dataset `D` and the oracle function to the `CLCB` function along with the delta for confidence.

```
import math
import numpy as np

# Simplified Oracle function: select action with the highest LCB
def oracle(LCBs):
    return np.argmax(LCBs)

# Generate a synthetic dataset D (m base arms, n samples)
# Each sample consists of a list of outcomes and the set of triggered
arms
m = 5  # Number of base arms
n = 10  # Number of samples

# Generate synthetic outcomes (random values between 0 and 1)
np.random.seed(42)  # For reproducibility
D = []
for _ in range(n):
```

```
    Xt = np.random.rand(m)  # Outcomes for each base arm
    triggered_arms = set(np.random.choice(m, size=np.random.randint(1,
m+1), replace=False))  # Random set of triggered arms
    D.append((Xt, triggered_arms))

# Print the synthetic dataset
print("Synthetic Dataset D:")
for i, (Xt, triggered_arms) in enumerate(D):
    print(f"Sample {i+1}: Outcomes {Xt}, Triggered arms
{triggered_arms}")

# CLCB Algorithm
def CLCB(D, oracle, delta):
    m = len(D[0][0])  # number of arms (based on the outcome size)
    n = len(D)  # number of samples
    LCBs = []

    for i in range(m):  # For each base arm
        Ni = sum([1 for t in range(n) if i in D[t][1]])  # Count how
many times arm i is triggered
        mu_hat_i = sum([D[t][0][i] for t in range(n) if i in D[t][1]])
/ Ni  # Empirical mean
        LCB_mu_i = mu_hat_i - math.sqrt(math.log(4 * m * n / delta) /
(2 * Ni))  # LCB for arm i
        LCBs.append(LCB_mu_i)

    # Step 2: Use oracle to select action
    S_hat = oracle(LCBs)

    return S_hat  # Return the selected combinatorial action

# Run the CLCB function with the dataset D
delta = 0.05  # Confidence level (probability of failure)
selected_action = CLCB(D, oracle, delta)

# Output the selected action (the arm with the highest LCB)
print(f"\nSelected combinatorial action: {selected_action} (arm with
the highest LCB)")
```

```
Synthetic Dataset D:
Sample 1: Outcomes [0.37454012 0.95071431 0.73199394 0.59865848
0.15601864], Triggered arms {0, 1, 3}
Sample 2: Outcomes [0.14286682 0.65088847 0.05641158 0.72199877
0.93855271], Triggered arms {1, 2}
Sample 3: Outcomes [0.61165316 0.00706631 0.02306243 0.52477466
0.39986097], Triggered arms {0, 1, 2, 4}
Sample 4: Outcomes [0.09060643 0.61838601 0.38246199 0.98323089
0.46676289], Triggered arms {0, 1, 2, 3, 4}
Sample 5: Outcomes [0.06505159 0.94888554 0.96563203 0.80839735
0.30461377], Triggered arms {0, 1, 2, 3, 4}
```

```
Sample 6: Outcomes [0.12203823 0.49517691 0.03438852 0.9093204
0.25877998], Triggered arms {0, 2, 3, 4}
Sample 7: Outcomes [0.54671028 0.18485446 0.96958463 0.77513282
0.93949894], Triggered arms {0, 1, 2, 4}
Sample 8: Outcomes [0.84453385 0.74732011 0.53969213 0.58675117
0.96525531], Triggered arms {0, 1, 2, 4}
Sample 9: Outcomes [0.80219698 0.07455064 0.98688694 0.77224477
0.19871568], Triggered arms {1, 3, 4}
Sample 10: Outcomes [0.77127035 0.07404465 0.35846573 0.11586906
0.86310343], Triggered arms {0, 2, 3, 4}

Selected combinatorial action: 3 (arm with the highest LCB)
```

1. **Dataset `D`**: The dataset consists of 10 samples, where each sample contains 5 base arms (outcomes) and a set of triggered arms (randomly selected for each sample). Each sample is represented as a tuple where the first element is a list of outcomes for each base arm (random values) and the second element is a set of triggered arms.
2. **Oracle**: The oracle simply selects the action with the highest LCB, which is computed for each base arm.
3. **LCB Calculation**: For each base arm, the empirical mean is calculated based on how often the arm was triggered across all samples, and the LCB is computed by subtracting a confidence term. That means: compute the empirical mean (average) of the outcomes and subtract the lower confidence bound term. The arm with the highest LCB is selected by the oracle.