**Summarized parts of linear Multiarmed bandit**

**1. Background and Problem Setup**

**Multi-Armed Bandits (MAB)**

The **Multi-Armed Bandit (MAB)** problem is a classic framework in machine learning and decision theory that models the trade-off between exploration (gathering information about different options) and exploitation (using known information to maximize rewards). Imagine a gambler facing multiple slot machines (arms), each with an unknown probability of payout. The goal is to maximize the total reward over a series of pulls.

**Linear Bandits**

**Linear Bandits** extend the MAB framework by assuming that the expected rewards of arms are linear functions of known feature vectors. This structure allows for the exploitation of correlations among arms, potentially leading to more efficient learning and decision-making.

**Best Arm Identification (BAI)**

While traditional MAB focuses on maximizing cumulative rewards, **Best Arm Identification (BAI)** aims to identify the arm with the highest expected reward with high confidence, typically under constraints like a fixed budget of arm pulls.

**2. Linear Bandits Overview**

**Mathematical Formulation**

In the **linear bandit** setting:

* **Arms**: There are KKK arms. Each arm iii is associated with a known feature vector a(i)∈Rda(i) \in \mathbb{R}^da(i)∈Rd.
* **Unknown Parameter**: There's an unknown parameter vector θ∗∈Rd\theta^\* \in \mathbb{R}^dθ∗∈Rd governing the expected rewards.
* **Rewards**: When arm iii is pulled at time ttt, a noisy reward is observed:

Xt=⟨θ∗,a(i)⟩+ηtX\_t = \langle \theta^\*, a(i) \rangle + \eta\_tXt​=⟨θ∗,a(i)⟩+ηt​

where ηt\eta\_tηt​ is zero-mean noise (often assumed to be Gaussian).

**Expected Rewards**

The expected reward for arm iii is:

μ(i)=⟨θ∗,a(i)⟩\mu(i) = \langle \theta^\*, a(i) \rangleμ(i)=⟨θ∗,a(i)⟩

The goal is to identify the arm i∗i^\*i∗ with the highest μ(i)\mu(i)μ(i):

i∗=arg⁡max⁡i∈{1,…,K}μ(i)i^\* = \arg\max\_{i \in \{1, \dots, K\}} \mu(i)i∗=argi∈{1,…,K}max​μ(i)

**3. Best Arm Identification (BAI)**

**Fixed-Budget Setting**

In the **fixed-budget** setting, the algorithm is given a total budget TTT (number of arm pulls) and must allocate these pulls across arms to maximize the probability of correctly identifying i∗i^\*i∗.

**Challenges**

* **Correlation Among Arms**: In linear bandits, arms are not independent; information from pulling one arm can inform estimates about others due to shared feature vectors.
* **Efficient Allocation**: Deciding how to allocate pulls to balance exploration and exploitation to accurately estimate θ∗\theta^\*θ∗ and, consequently, i∗i^\*i∗.

**4. Optimal Design Principles**

**Experimental Design**

**Optimal Design** is a field in statistics that focuses on selecting the most informative experiments to efficiently estimate model parameters. In the context of linear bandits, it involves choosing how often to pull each arm to minimize uncertainty in estimating θ∗\theta^\*θ∗.

**G-Optimal Design**

A specific type of optimal design, **G-Optimal Design**, aims to minimize the maximum prediction variance across all possible arms. This ensures that the worst-case uncertainty is as low as possible, which is crucial for reliably identifying the best arm.

**Connection to OD-LinBAI**

The OD-LinBAI algorithm leverages principles from optimal design, particularly G-Optimal Design, to allocate the fixed budget of arm pulls in a way that minimizes uncertainty in estimating the expected rewards, thereby increasing the probability of correctly identifying the best arm.

**5. OD-LinBAI Algorithm Steps**

The OD-LinBAI algorithm operates in **phases**, iteratively refining the set of candidate arms and allocating pulls based on optimal design principles. Here's a step-by-step breakdown:

**Initialization**

1. **Active Set AAA**: Start with all arms A={1,2,…,K}A = \{1, 2, \dots, K\}A={1,2,…,K}.
2. **Phase Count**: Initialize the phase counter r=1r = 1r=1.

**Iterative Phasing**

For each phase rrr, perform the following steps:

**Step 1: Optimal Design Computation**

* **G-Optimal Design πr\pi\_rπr​**: Compute an (ε-approximate) G-optimal design over the current active set Ar−1A\_{r-1}Ar−1​.
  + This design determines the proportion of arm pulls to allocate to each arm in Ar−1A\_{r-1}Ar−1​ to minimize the maximum prediction variance.

**Step 2: Arm Pull Allocation**

* **Number of Pulls TrT\_rTr​**: Determine the number of arm pulls for phase rrr based on the budget and phase-specific parameters.
* **Allocate Pulls**: Distribute TrT\_rTr​ pulls among the arms in Ar−1A\_{r-1}Ar−1​ according to πr\pi\_rπr​.

**Step 3: Estimation**

* **Estimate Rewards**: For each arm iii in Ar−1A\_{r-1}Ar−1​, estimate the expected reward μ^r(i)\hat{\mu}\_r(i)μ^​r​(i) using the collected data.
  + Typically, this involves Ordinary Least Squares (OLS) estimation:

θ^r=Vr−1∑t=1Tra(At)Xt\hat{\theta}\_r = V\_r^{-1} \sum\_{t=1}^{T\_r} a(A\_t) X\_tθ^r​=Vr−1​t=1∑Tr​​a(At​)Xt​

where Vr=∑t=1Tra(At)a(At)⊤V\_r = \sum\_{t=1}^{T\_r} a(A\_t) a(A\_t)^\topVr​=∑t=1Tr​​a(At​)a(At​)⊤ is the design matrix.

* + Then,

μ^r(i)=⟨θ^r,a(i)⟩\hat{\mu}\_r(i) = \langle \hat{\theta}\_r, a(i) \rangleμ^​r​(i)=⟨θ^r​,a(i)⟩

**Step 4: Elimination**

* **Candidate Set Update**: Based on the estimates μ^r(i)\hat{\mu}\_r(i)μ^​r​(i), eliminate a subset of arms that are unlikely to be the best.
  + Typically, eliminate the bottom fraction of arms with the lowest estimated rewards.
  + Update the active set ArA\_rAr​ accordingly.

**Step 5: Termination Check**

* **Stopping Condition**: Check if the algorithm has reached the maximum number of phases or if only one arm remains in ArA\_rAr​.
  + If termination criteria are met, output the remaining arm as iouti\_{\text{out}}iout​.
  + Otherwise, increment rrr and proceed to the next phase.

**Final Output**

After completing all phases or upon early termination, the algorithm outputs iouti\_{\text{out}}iout​, the arm identified as the best based on the accumulated data and elimination steps.

**6. Theoretical Guarantees**

OD-LinBAI provides theoretical guarantees on the probability of correctly identifying the best arm within the fixed budget TTT. Key aspects include:

**Error Probability Bound**

The probability that the algorithm fails to identify the best arm i∗i^\*i∗ is bounded by:

Pr⁡[iout≠i∗]≤(4Kd+3log⁡2d)exp⁡(−m32H2,lin)\Pr[i\_{\text{out}} \neq i^\*] \leq \left( \frac{4K}{d} + 3 \log\_2 d \right) \exp\left( -\frac{m}{32 H\_{2,\text{lin}}} \right)Pr[iout​=i∗]≤(d4K​+3log2​d)exp(−32H2,lin​m​)

where:

* KKK is the number of arms.
* ddd is the dimensionality of the feature vectors.
* mmm is a parameter related to the budget allocation.
* H2,linH\_{2,\text{lin}}H2,lin​ captures the hardness of the instance, defined as: H2,lin=max⁡2≤i≤d1Δi2H\_{2,\text{lin}} = \max\_{2 \leq i \leq d} \frac{1}{\Delta\_i^2}H2,lin​=2≤i≤dmax​Δi2​1​ with Δi\Delta\_iΔi​ being the gap between the best arm and arm iii.

**Near-Optimal Design**

By utilizing near-optimal designs with smaller support (i.e., fewer arms actively being considered), OD-LinBAI maintains efficiency in high-dimensional settings, ensuring that the algorithm remains computationally feasible even as ddd grows.

**Phase-wise Analysis**

The algorithm's phased approach allows for iterative refinement of the active set, with theoretical bounds ensuring that the probability of eliminating the best arm decreases exponentially with the budget allocated per phase.

**7. Practical Considerations and Applications**

**Computational Efficiency**

* **Frank–Wolfe Algorithm**: OD-LinBAI employs the Frank–Wolfe algorithm to compute approximate G-optimal designs efficiently, ensuring scalability to larger problem instances.
* **Support Size**: By utilizing designs with smaller support, the algorithm reduces computational overhead, especially beneficial in high-dimensional settings.

**Applications**

* **Online Advertising**: Selecting the most effective ad to display based on contextual features.
* **Clinical Trials**: Identifying the most promising treatment among multiple options based on patient data.
* **Recommendation Systems**: Determining the best product to recommend to users based on their interaction history and product features.

**Assumptions and Limitations**

* **Linear Reward Structure**: Assumes that expected rewards are linear in the feature vectors, which may not hold in all real-world scenarios.
* **Fixed Budget**: The algorithm is designed for fixed-budget settings; adapting it to other settings (e.g., fixed confidence) may require modifications.
* **Noise Assumptions**: Performance guarantees often rely on assumptions about the noise (e.g., Gaussian), and deviations from these assumptions can affect performance.
* **Summary**
* The **OD-LinBAI algorithm** is a powerful method for identifying the best arm in linear bandit settings under fixed-budget constraints. By leveraging optimal design principles, particularly G-optimal designs, and employing a phased elimination strategy, OD-LinBAI efficiently allocates resources to maximize the probability of correct identification. Its theoretical guarantees and practical efficiency make it a valuable tool in various applications where decision-making under uncertainty and resource constraints is crucial.