Implementing Efficient Linear Bandits via Sketches and Random Projections

Lilia Izri
LISN, CNRS
Université Paris-Saclay
Orsay, France
izri@lisn.fr

Benoît Groz
LISN, CNRS
Université Paris-Saclay
Orsay, France
groz@lisn.fr

Silviu Maniu
Univ. Grenoble Alpes, CNRS,
Grenoble INP, LIG
Grenoble, France
silviu.maniu@univ-grenoble-alpes.fr

***Abstract*—Bandit algorithms, particularly linear bandits, are highly effective at balancing exploration and exploitation to learn models that are much simpler than full-scale deep learning architectures. They are useful, among others, for recommender systems, influence maximization, and routing. However, and especially for linear bandits, the computational time and storage of these algorithms increase quadratically with the dimension of input data, making them relatively expensive to run in practice, especially on low-powered devices.**

Theoretical approaches aiming to optimize the computation have been proposed, based on dimension- and data-reduction techniques, such as sketches and random projections. In this article, we implement and evaluate experimentally how these dimensionality reduction techniques can help reduce the time and space complexity while keeping recommendation performance high and bandit regret low. To this end, we compare on several real-world datasets existing approaches in the literature along with a new adaptation of bandits based on random projections. Our results show that random projections are very efficient at reducing the execution time and memory footprint, whereas sketching is very efficient at reducing the cumulative regret, and that both can contribute to faster low-regret implementations of linear multi-armed bandits. These findings are particularly relevant to Big-Data deployments, where the high dimensionality of features makes trade-offs between actual performance, strict latency and memory constraints essential for practical implementation.

***Index Terms*—Sequential Algorithms, Recommendations, Linear Multi-Armed Bandits, Dimensionality Reduction, Sketching, Random Projections.**

I. Introduction

Multi-Armed Bandit (MAB) algorithms, first proposed by [1], are simple and efficient algorithms that have proven popular for applications that require sequential decision-making on uncertain inputs. However, time and memory requirements of these algorithms can sometimes be prohibitive in practice on high dimensional data. Particularly, in constrained environments like Internet of Things (IoT) devices that are characterized by small storage capacity and power constraints, the demands of some of these algorithms can exceed the available resources, causing a gap between theoretical and actual performance. Consequently, developing resource-efficient algorithms is essential to enable practical application in real-world scenarios [2], [3]. This paper investigates the impact of dimensionality reduction techniques on linear bandit algorithms by providing a comprehensive implementation, evaluation, and analysis of their performance, time complexity, and memory usage across synthetic and real-world datasets, and providing solutions for practical efficiency in recommendation applications.

The MAB problem involves deciding which actions to take over time to maximize cumulative rewards, without any initial information concerning the *reward* associated with each action (or *arm*). The MAB problem can equivalently be formulated as a problem of minimizing the *regret*, defined as the cumulative difference between the optimal reward achievable by consistently selecting the best action and the reward gained by the algorithm. [4] have proved that no algorithm can obtain a regret bound better than $\Omega(\sqrt{T})$ where T is the number of iterations; a sub-linear regret bound essentially means that the learning rate is faster compared to basic strategies. Bandit algorithms dynamically optimize the decision-making process by efficiently balancing *exploration*, where options are explored to gain information and learn (simple) models, and *exploitation*, where the learned information is used to maximize rewards.

MAB algorithms based on the Upper Confidence Bound (UCB) principle [5], [6] offer strong theoretical guarantees [4]. The UCB principle involves balancing between exploration and exploitation by selecting the action with the highest UCB, thus favoring the exploitation of less-explored actions that have high uncertainty. *Linear Bandits* (LB) extends the MAB framework by introducing d-dimensional vectorized arms and linear rewards defined as the scalar projection of the selected arm on a model parameter vector θ_*. LB algorithms exploit this linear reward structure to converge to optimal solutions with few samples.

LinREL [4] is one of the first algorithms to adapt the UCB principle to LB. This algorithm uses eigenvalue decomposition to estimate the parameter θ_* and achieves an optimal regret upper bound $\tilde{O}(\sqrt{dT})$. Variants and analyses of this algorithm have since been proposed to improve regret bounds, as in the OFUL algorithm [7].

Bandit algorithms have been applied in multiple settings [8], such as routing, influence maximization, optimal medical usage, and cell planning in 5G networks. Here, we focus on **recommendation systems** applications. Recommendations

are a natural fit for MABs: actions (arms) correspond to the vectorial representations of recommended items, whereas user interactions with these items determine rewards. For example, studies such as [9]–[11] applied bandits on the Yahoo and MovieLens datasets to recommend news articles and movies. The approaches that we present in this paper generalize naturally to other applications.

The complexity of estimating the parameter vector of linear bandits grows quadratically with its dimensionality, for both time and storage requirements, making their application costly on real-world datasets having large dimensions. Various dimensionality reduction methods can be employed to mitigate this complexity and enable the handling of large-scale datasets with limited computational resources. These techniques aim to reduce high-dimensional data into lower-dimensional representations while preserving the essential information needed to minimize regret.

In this paper, we study practical implementation aspects and the impact of dimensionality reduction techniques on the performance of linear bandit algorithms for recommendation. We cover the following:

1) We **discuss and compare the two main approaches to dimensionality reduction** (sketching and random projections), and introduce an adaptation of the linear bandit algorithm CONFIDENCEBALL1, which leverages random projections to reduce the time and memory requirements.
2) We conduct a **comprehensive evaluation of the algorithms** on synthetic and real-world datasets, comparing their performance (recommendation Click-Through-Rate "CTR" and regret), time complexity, and memory complexity. Our objective is to provide an understanding of the advantages and potential limitations of each approach, for potential use for practitioners.
3) We provide **ready-to-use implementations of the studied approaches**, available online at https://github.com/LiliZr/Stochastic-linear-bandits.

Paper outline: In Section II, we recall two well-known linear stochastic bandit algorithms, detailing their underlying principles and theoretical regret guarantees. Section III presents and compares bandit algorithms that leverage dimensionality reduction techniques, including our proposed approach, highlighting their respective advantages and limitations. Section IV reviews additional relevant literature. Finally, Sections V and VI present an empirical evaluation of the algorithms on synthetic and real-world datasets, followed by a discussion of the results.

II. LINEAR MULTI-ARMED BANDITS

In MAB problems, the agent (decision maker) iteratively (i) receives a set of actions, (ii) selects one of them, and (iii) observes the resulting reward. Bandits can be used as recommendation systems where the objective is to efficiently balance the exploration-exploitation process in order to maximize the cumulative expected reward. Stochastic linear bandits can be seen as a generalization of the bandit problem where the actions form a set of vectors $\mathcal{A} \subset \mathbb{R}^d$ and the reward obtained after choosing an action from $\mathcal{A}$ is the dot product of that action with an unknown fixed parameter $\theta_* \in \mathbb{R}^d$, plus a small noise η. Moreover, the action set may vary over time, resulting in a contextual bandit problem where the action set at iteration t is denoted by $\mathcal{A}_t \subset \mathcal{A}$. Table I summarizes the notations used in this paper.

TABLE I: Notation used in the paper.

T	Maximum iterations
d	Original dimension
n	Number of actions (items)
m	Sketching dimension
k	Projection dimension
λ	Regularization factor of $l2-$least squares
η	Noise added to the reward $\eta \sim \mathcal{N}(0, \sigma^2)$
a_t	Selected action at time t
X_t	List of selected actions $X_t = [a_1, a_2, ..., a_t]$
V_t	Covariance matrix
θ_*	Optimal parameter vector
r_t	Reward observed at time t
$\hat{R}_T$	Cumulative estimated regret at time T
S_t	Sketched selected actions list
$\tilde{V}_t$	Sketched covariance matrix
ε	Projection error or distortion $\in (0, 1)$
Φ	$k \times d$ projection matrix
x_a	Projected action Φa
μ_*	Projected model parameter $\Phi\theta_*$
$[T]$	Set from 1 to T
$\hat{x}$	Estimator of parameter x
$\|x\|_{1,V}$	$\|A^{1/2}x\|_1$
$\|x\|_{2,V}$	$\sqrt{x^\top V x}$

A. Setting

We consider a linear bandit algorithm with a finite set $\mathcal{A}$ of n actions (or items) of dimension d, and an unknown parameter $\theta_* \in \mathbb{R}^d$. At each step t, the system recommends an action $a_t \in \mathbb{R}^d$ and receives a reward r_t having a linear form, such that:

$$r_t = \langle \theta_*, a_t \rangle + \eta.$$

Remark 1: We assume that η follows a Gaussian distribution $\eta \sim \mathcal{N}(0, \sigma^2)$. In addition, we will assume that $\|\theta_*\|_2 \leq 1$ and $\|a\|_2 \leq 1, \forall a \in \mathcal{A}$. Hence, $|\langle \theta_*, a_t \rangle| \leq 1$ for all $a_t \in \mathcal{A}$.

The goal is to select actions to maximize the expected cumulative reward over a specified time horizon T. Maximizing the cumulative reward is equivalent to minimizing the regret R_T, defined as the difference at time T between the expectations for the maximum cumulative reward achievable by consistently selecting the best action at each step t and the actual cumulative reward we obtain for the sequence of actions selected by the algorithm. The expected regret can be written as follows:

$$R_T = \mathbb{E}\left[\sum_{t=1}^{T} \max_{a \in \mathcal{A}} \langle \theta_*, a \rangle - \sum_{t=1}^{T} \langle \theta_*, a_t \rangle\right].$$

No algorithm can guarantee a regret bound smaller than $\Theta(\sqrt{T})$ achieved in [4]. Generally, a strategy balancing exploration and exploitation is considered efficient when regret is sublinear in T.

B. Linear Bandit Algorithms

In this section, we present two well-known baseline linear bandit algorithms with optimal performance in terms of regret minimization [12], [13]. Both implement the UCB strategy, which, based on the principle of "optimism in the face of uncertainty", guides the algorithm to estimate an optimistic reward prediction as the upper bound of some confidence interval, and then select the action with the highest estimation. In the case of linear rewards, this principle is adapted by constructing a confidence set $\mathcal{C}_t \subset \mathbb{R}^d$ that contains the optimal model parameter $\theta_* \in \mathbb{R}^d$. Then, it defines an optimistic guess for the reward of each action as $\mathrm{UCB}(a) = \max_{\theta \in \mathcal{C}_t} \langle \theta, a \rangle$ and recommends the action corresponding to the highest guess: $a_t = \mathrm{argmax}_a \, \mathrm{UCB}(a)$. After that, the algorithm observes a reward $r_t = \langle \theta_*, a_t \rangle + \eta$, and estimates the model parameter by solving:

$$\hat{\theta}_t = \mathrm{argmin}_{\theta \in \mathbb{R}^d} \left(\sum_{s=1}^{t} \left(r_s - \langle \theta, a_s \rangle \right)^2 + \lambda \|\theta\|_2^2 \right),$$

where λ is a regularization parameter.

In order to compute $\hat{\theta}_t$, the algorithm maintains the $d \times d$ covariance matrix $V_t = \lambda I + \sum_{s=1}^{t} a_s a_s^\top$. The closed form equation of $\hat{\theta}_t$ is:

$$\hat{\theta}_t = V_t^{-1} \sum_{s=1}^{t} a_s \cdot r_s.$$

We can construct a confidence set at time t, which is an ellipsoid $\mathcal{C}_t$ centered at $\hat{\theta}_{t-1}$ (the estimator of θ_* at time $t-1$). With probability $1-\delta$, for each t, θ_* belongs to:

$$\mathcal{C}_t = \left\{ \theta \in \mathbb{R}^d : \left\| \theta - \hat{\theta}_{t-1} \right\|_{p, V_{t-1}} \leq \sqrt{\alpha_p \beta_t} \right\},$$

where $p \in \{1, 2\}$ corresponds to the L1 or L2 norm, $\alpha_1 = 1, \alpha_2 = d$ and $(\beta_t)_t$ is an increasing sequence, s.t. $\sqrt{\beta_t} = \sqrt{\lambda} + \sqrt{2 \log(\frac{1}{\delta}) + d \log\left(1 + \frac{t}{d\lambda}\right)}$ [14].

The two algorithms diverge in their approach for computing the UCB of an action. [13] introduces the LINUCB algorithm for the case of a finite set of actions where the arm with the highest UCB is chosen s.t. $a_t = \mathrm{argmax}_{a \in \mathcal{A}} \langle \hat{\theta}_{t-1}, a \rangle + \sqrt{\beta_t} \|a\|_{V_{t-1}^{-1}}$. As this proposed strategy is technically difficult to analyze, the regret is analyzed only on a modified version of the algorithm where the regret is bounded by $\tilde{O}\left(\sqrt{dT}\right)$. [12] introduces two versions of an algorithm named CONFIDENCEBALL (or CBALL) that can handle both the case of a finite action set as well as the infinite case. CONFIDENCEBALL2 maintains a confidence set using the L2-norm leading to better regret guarantees. CONFIDENCEBALL1 approximates the confidence ellipsoid with an L1 polytope and thus runs in polynomial time. The use of L2-ellipsoid leads to a regret bound similar to the one of LINUCB, while using the L1-ellipsoid leads to a slightly worse bound of $\tilde{O}\left(d^{\frac{3}{2}}\sqrt{T}\right)$.

Despite their optimal performance in terms of regret, these algorithms estimate the unknown model parameter θ_* using a regularized linear regression, which is challenging on large real-world datasets. Specifically, computing the required estimator implies inverting a matrix of dimension d, which over the course of the algorithm costs $O(d^2)$ in memory and $O(d^2 T)$ in total time when using the Sherman-Morrison formula, which is the fastest method for a rank-1 update of the inverse matrix. This quadratic dependency on the original dimensions makes these algorithms impractical for high-dimensional cases.

III. DIMENSIONALITY REDUCTION FOR LINEAR MABS

Research has focused on improving the time complexity of baseline LB algorithms, using matrix projection and sketching techniques similar to the ones adopted for linear regression. These techniques reduce computational complexity by projecting high-dimensional input data (actions) onto lower-dimensional spaces or by sketching matrices to approximate complex computations. In this section, we present the various existing algorithms by outlining the technical challenges to keep regret low.

A. Sketching the Covariance Matrix

Frequent Directions (FD) [15] is one of the most effective dimensionality reduction methods, especially for sequential data. This deterministic method takes as input a stream of rows from a matrix $A \in \mathbb{R}^{t \times d}$ and refines a sketch matrix $B \in \mathbb{R}^{m \times d}$ with $m \ll t$ to approximate the principal components of A.

To date, two distinct linear bandit algorithms have been developed using this method [16], [17]. Both algorithms apply frequent directions to a UCB bandit algorithm [7], [13]. Algorithm 1 details the fundamental template common to those two algorithms. The template has been simplified to consider a fixed action set instead of one that depends on the step (contextual bandit).

Algorithm 1 Linear Bandits – Sketching the Covariance Matrix

Require: Action set $\mathcal{A} \subset \mathbb{R}^d$, $\lambda > 0$, $\sigma \in [0, 1]$

Initialization:
  $\tilde{\theta}_0 = 0_d$, $S_0 = 0_{m \times d}$, $H_0 = \frac{1}{\lambda} I_{m \times m}, \alpha_0 = \lambda$
for $t \in [T]$ **do**
  Compute $\sqrt{\beta_t}$
  $a_t = \mathrm{argmax}_{a \in \mathcal{A}} \langle \tilde{\theta}_{t-1}, a \rangle + \sqrt{\beta_t} \|a\|_{\tilde{V}_{t-1}^{-1}}$ ▷*Highest UCB arm*
  Observe the reward $r_t = \langle \theta_*, a_t \rangle + \eta$ ▷*With* $\eta \sim \mathcal{N}(0, \sigma^2)$
  UPDATEPARAMETERS$(S_t, H_t, a_t, \alpha_{t-1})$ ▷*FD Update [16], [17]*
  $\alpha_t \leftarrow \alpha_{t-1}$
  $\tilde{V}_t^{-1} = \frac{1}{\alpha_t} \left(I_{d \times d} - S_t^\top H_t S_t \right)$
  $\tilde{\theta}_t = \tilde{V}_t^{-1} \sum_{s=1}^{t} a_s r_s$ ▷*Update estimator (linear regression)*
end for

The sketch maintains an approximation $S_t \in \mathbb{R}^{m \times d}$ of $X_t = [a_1, a_2, ..., a_t]$, with $m \ll \min(t, d)$. The covariance matrix

$V_t = X_t^\top X_t + \lambda I$ is approximated by $\tilde{V}_t = S_t^\top S_t + \lambda I$. The inverse of the covariance matrix is $\tilde{V}_t^{-1} = \frac{1}{\lambda}\left(I_{d\times d} - S_t^\top H_t S_t\right)$ with S_t the sketching matrix that approximates the row space of the original matrix X_t by retaining the top m singular vectors, thereby capturing the most significant directions of X_t and $H_t = \left(S_t S_t^\top - \lambda I_{m\times m}\right)^{-1}$. For a more detailed explanation of the computation of these matrices, refer to [16], [17].

In practice, neither $\tilde{V}_t$ nor $\tilde{V}_t^{-1}$ are fully materialized because all operations involving $\tilde{V}_t^{-1}$ are matrix-vector multiplications. The new estimator of the model parameter $\tilde{\theta}_t$ is then computed as $\tilde{\theta}_t = \tilde{V}_t^{-1}\sum_{s=1}^{t} a_s \cdot r_s$.

SOFUL*:* The SOFUL algorithm [16] uses the frequent directions method [15] to transform the set of t actions into a smaller representation of size m (where $m \ll t$). Sketching the correlation matrix V_t using FD reduces the total time complexity from $O(d^2T)$ to $O(mdT)$ and the memory complexity from $O(d^2)$ to $O(md)$ while keeping a sub-linear regret upper bound of $\tilde{O}\big((1+\Delta_T)^{\frac{3}{2}}(m + d\log(1+\Delta_T))\sqrt{T}\big)$ where Δ_T is upper bounded by the spectral tail, i.e., the sum of the smallest $d-m+1$ eigenvalues of the correlation matrix. However, if the chosen m is too small (spectrum with heavy tail or m smaller than the rank of X_t), this algorithm suffers linear regret. SOFUL can be viewed as Algorithm 1 where line 4 is adapted to compute $\sqrt{\beta_t}$ as $\sqrt{\beta_t} = \sqrt{m\ln\big(1+\frac{t}{m\lambda}\big) + 2\ln\big(\frac{1}{\delta}\big) + d\ln\big(1+\frac{\bar{\rho}_t}{\lambda}\big)}\cdot\sqrt{1+\frac{\bar{\rho}_t}{\lambda}} + \sqrt{\lambda}\left(1+\frac{\bar{\rho}_t}{\lambda}\right)$, having $\bar{\rho}_t = \rho_1 + \cdots + \rho_t$ where ρ_t is the smallest eigenvalue of the sketch $S_t^\top S_t$.

CBSCFD*:* [17] introduces CBSCFD, an algorithm similar to SOFUL, but where the information lost in FD is compensated by adding a diagonal matrix containing spectral information that would otherwise be discarded by the approximations from the original FD sketching. This reduces the impact of Δ_T on the regret bound, leading to a lower regret bound than the one from SOFUL: $\tilde{O}\big(\sqrt{mT}\big(\sqrt{m + d\log(1+\Delta_T)} + \sqrt{\Delta_T}\big)\big)$. The time and space complexities of CBSCFD are the same as those of SOFUL. CBSCFD is obtained by using the following value in line 4 of Algorithm 1: $\sqrt{\beta_t} = \sqrt{\lambda+\Delta_t} + \sqrt{2\ln\big(\frac{1}{\delta}\big) + m\ln\big(1+\frac{t}{m\lambda}\big) + d\ln\big(1+\frac{\Delta_t}{\lambda}\big)}$ where $\Delta_t = \sum_{i=1}^{t}\delta_t$. Moreover, line 8 in Algorithm 1 must be replaced by $\alpha_t \leftarrow \alpha_{t-1} + \delta_t$ where δ_t is the m-th singular value of S_t.

Impact of FD Sketching on Bandit Algorithms: The Frequent Directions (FD) technique provides an effective and deterministic approach for linear bandits: employing FD to sketch the covariance matrix can yield sub-linear regret bounds in theory. However, its performance can be sensitive to the data distribution (e.g., action matrix having high rank or heavy-tailed spectrum distributions). This can lead to difficulties in capturing important information, which could result in an inaccurate approximation. Additionally, when using relatively high dimensional data, the computational cost of $O(md)$ is much higher than the technique we discuss next.

B. Projecting the Action Set

Johnson-Lindenstrauss transformations, such as random projections, are another way of reducing dimensionality. These methods attempt to efficiently compress data points from a high-dimensional space into a lower-dimensional one while preserving important information.

The Johnson–Lindenstrauss lemma [18] is a result stating that a set of points in a high-dimensional space can be embedded into a much lower-dimension space in such a way that distances between all pairs of points are preserved within a factor of $1 \pm \varepsilon$.

Importantly, such linear mappings can be computed efficiently.

In this section, we consider approaches that project the action set to a lower dimension using a matrix transformation $\Phi \in \mathbb{R}^{k\times d}$. After projection, the agent must learn the parameter $\mu_* = \Phi\theta_* \in \mathbb{R}^k$. The objective remains the same, minimizing the cumulative regret. However, as we are working in a reduced dimension, the estimation of μ_* is adapted by computing at each round t the estimator $\hat{\mu}_*$ s.t.:

$$\hat{\mu}_t = \operatorname{argmin}_{\mu\in\mathbb{R}^k}\left(\sum_{s=1}^{t}\left(r_s - \langle \mu, x_{a_s}\rangle\right)^2 + \lambda\|\mu\|_2^2\right), \quad (1)$$

where r_s is the reward obtained at time s, x_{a_s} is the projection of the action a_s using Φ and $\lambda > 0$ is a regularization factor.

Algorithm 2 Linear Bandits – Projection Matrix

Require: Action set $\mathcal{A} \subset \mathbb{R}^d,\ \lambda > 0,\ \sigma \in [0,1]$

1: **Initialization:**
2: Generate the projection matrix Φ
3: Project $\mathcal{A}$: compute $x_a = \Phi a, \quad \forall a \in \mathcal{A}$
4: $\hat{\mu}_0 = 0_k$, $V_0 = \lambda I_{k\times k}$
5: **for** $t \in [T]$ **do**
6: $a_t = \operatorname{argmax}_{a\in\mathcal{A}} \max_{\mu\in\mathcal{C}_t}\langle \mu, x_a\rangle$ ▷*Highest UCB arm*
7: Observe the reward $r_t = \langle \theta_*, a_t\rangle + \eta$ ▷*With* $\eta \sim \mathcal{N}(0,\sigma^2)$
8: $V_t = V_0 + \sum_{s=1}^{t} x_{a_s} x_{a_s}^\top$
9: $\hat{\mu}_t = V_t^{-1}\sum_{s=1}^{t} x_{a_s} r_s$ ▷*Update estimator (linear regression)*
10: **end for**

The general flow, illustrated in Algorithm 2, remains the same as in the baseline UCB-based Algorithms for a non-contextual case. At step t, the algorithm recommends the action a_t having the highest UCB using a confidence set $\mathcal{C}_t$ s.t. $\mathrm{UCB}(a) = \max_{\mu\in\mathcal{C}_t}\langle \mu, x_a\rangle$. Then, it observes a noisy linear reward r_t, updates the necessary parameters and computes the estimated model parameter using the closed form of Eq. 1.

The main difference to the original algorithm is that the projected actions (line 3), the action covariance matrix and the model parameter estimator are in dimension k.

CBRAP*:* CBRAP [19] adapts LinUCB [13] using a random projection matrix. Specifically, in Algorithm 2 line 2, a matrix $\Phi \in \mathbb{R}^{k\times d}$ is generated where each element follows a normal distribution $\mathcal{N}(0, 1/k)$. Then, it builds a confidence

set $\mathcal{C}_t = \{\mu \in \mathbb{R}^k : \|\mu - \hat{\mu}_{t-1}\|_{2,V_{t-1}} \leq \sqrt{\beta_t}\}$, where $\sqrt{\beta_t} = \sqrt{k \log \frac{1+t}{\delta}} + \sqrt{\lambda} + \epsilon\sqrt{t}$ and $\epsilon = \Omega(\sqrt{\frac{1}{k}})$. Finally, the action a_t is chosen in line 6: $a_t = \text{argmax}_{a \in \mathcal{A}_t} \langle \hat{\mu}_{t-1}, x_a \rangle + \sqrt{\beta_t} \|x_a\|_{V_{t-1}^{-1}}$.

By reducing the dimensionality, CBRAP thus lowers the time complexity of the algorithm from $O(d^2T)$ to $O(k^2T)$ and the memory complexity from $O(d^2)$ to $O(k^2)$. Nevertheless, this reduction is not without drawbacks. In particular, the random projection introduces an error term εT in the regret bound, resulting in near-linear regret $\tilde{O}\left(k\sqrt{T} + \varepsilon\sqrt{k}T\right)$. The less effective the projection dimension k (too small compared to the original dimension), the greater the impact of the introduced error term (i.e., larger error distortion $1 \pm \varepsilon$).

CONFIDENCEBALL1_FJLT*:* We introduce here a simple adaptation of CONFIDENCEBALL1 [7], [12] using the Fast Johnson-Lindenstrauss Transform (FJLT) [20]. FJLT is a transformation that respects the JL lemma but is faster than random projections. In particular, to implement line 2 in Algorithm 2, we generate a FJLT projection matrix $\Phi = \frac{1}{\sqrt{k}} PHD$ (with P, H and D as described in [20]), then we project our initial data points (actions) of dimension d, into a lower dimension k using Φ. Then, the equation in line 6 is solved by generating 2k extremal points of the polyhedron confidence set defined as $\mathcal{C}_t = \left\{\mu \in \mathbb{R}^k : \|\mu - \hat{\mu}_{t-1}\|_{1,V_{t-1}} \leq \sqrt{k\beta_t}\right\}$ with $\sqrt{\beta}_t = \sqrt{\lambda} + \sqrt{2\log(\frac{1}{\delta}) + k\log\left(1 + \frac{t}{k\lambda}\right)}$.

As in the CBRAP algorithm, the use of random projections here again introduces a term εT in the regret bound, resulting in a regret of $\tilde{O}\left(k\sqrt{T} + 2\varepsilon T\right)$. The time and memory complexities of this algorithm are the same as the ones of CBRAP.

Impact of Random Projections on Bandit Algorithms: Using projection matrices offers faster computation compared to other techniques. After the projection, the computation time no longer depends on original dimension d. Furthermore, the algorithm involves only matrix multiplications which is faster than PCA or other decompositions (especially in contextual settings when the action set changes over time). On the other hand, the guarantees of projections are not deterministic.

Table II summarizes the regret, time and space complexities of the algorithms discussed in this section.

IV. OTHER RELATED WORK

We briefly survey other approaches for efficient bandit algorithms.

[14] provides a comprehensive overview of various bandit algorithms and principles. [21] presents a novel algorithm called LRP for sparse linear bandit problems that combines a lasso estimator and adaptive random projections. The regret upper bound for LRP-Bandit is $\tilde{O}(s\sqrt{T \log d})$ where $s \ll d$ is the number of features that have non-zero coefficient values in θ_*. However, implementing this algorithm efficiently is a challenging task due to the use of complex convex optimizations. [22] proposes PSLB, a PCA-based method that learns a low-dimensional subspace of the action set. Although PCA has been shown to yield more informative projections than data-independent transforms (such as random projections), it requires additional computational operations and may lead to higher running time. Another approach in the literature combines projections to reduce time complexity and kernel methods to handle situations where the reward function may be complex and nonlinear: the BKB [23] and EK-UCB [24] algorithms propose two different Gaussian process based algorithms for kernel bandits combined to Nyström projections. Both these algorithms achieve a regret $\tilde{O}(\sqrt{T}(d_{\text{eff}} + \sqrt{\lambda d_{\text{eff}}}))$ where d_{eff} is the effective reduced dimension as described in [25]. The time complexity of BKB is $O(T^2 d_{\text{eff}}^2 + nTd_{\text{eff}}^2)$ while EK-UCB has a time complexity of $O(nTd_{\text{eff}}^2)$. This type of algorithm using kernel methods appears to have better theoretical results in terms of regret, yet they tend to be slower in practice than algorithms presented in Section III. [26] presents the D-LINTS-RP algorithm, which adapts the Thompson Sampling algorithm [27] for non-stationary environments with random projections. The D-LINTS-RP algorithm achieves a regret bound $\tilde{O}\left(T(\exp(-k\varepsilon^2) + \varepsilon(1 + \sqrt{\frac{k}{T}\log(n)}))\right)$ with ε the distortion error of random projections and $k < d$ the projected dimension.

V. EXPERIMENTAL SETUP AND RESULTS

In this section, we first present the experimental setup: the datasets we used, the implementation details, and the parameter space for evaluation. We then present the results obtained for the different datasets and algorithms tested.

A. Datasets and Settings

Dataset description and processing: To conduct the experiments, we used two synthetic datasets—Synthetic 1 (n=200, d=100) and Synthetic 2 (n=2000, d=1000)— where each action and θ_* are drawn i.i.d. from $\mathcal{N}(0, 1)$ (n = number of actions, d = dimension). We also evaluate the algorithms on five real-world datasets: MNIST[1], Steam[2], MovieLens 25M[3], Amazon Books[4], Yahoo[5].

The bandit algorithms are used to provide recommendations: actions are the vector representations of items, obtained by concatenating available features with the multi-label encoding of users who interacted with them, and the reward corresponds to the adoption of the recommended item. For real-world datasets, we retain only the most relevant users, i.e., those who have interacted with a sufficient number of items—to ensure a realistic estimation of θ_* within a reasonable execution time. Full dataset details and preprocessing steps are available online.[6]

[1]https://pjreddie.com/projects/mnist-in-csv/

[2]https://kaggle.com/datasets/antonkozyriev/game-recommendations-on-steam

[3]https://grouplens.org/datasets/MovieLens/

[4]https://amazon-reviews-2023.github.io/data_processing/index.html

[5]https://webscope.sandbox.yahoo.com/catalog.php?datatype=r&did=49

[6]https://github.com/LiliZr/Stochastic-linear-bandits

TABLE II: Comparison of regret and time and space complexities of the algorithms.

Algorithm	Regret	Time	Space
Baseline algorithms			
CONFIDENCEBALL1 [12]	$\tilde{O}\left(d^{\frac{3}{2}}\sqrt{T}\right)$	$O(nd^2T)$	$O(nd^2)$
LINUCB [13]	$\tilde{O}\left(\sqrt{dT}\right)$	$O(nd^2T)$	$O(nd^2)$
Covariance Matrix Sketching			
SOFUL [16]	$\tilde{O}\left(\sqrt{T}\,(1+\Delta_T)^{\frac{3}{2}} \cdot (m + d\log(1+\Delta_T))\right)$	$O(nmdT)$	$O(nmd)$
CBSCFD [17]	$\tilde{O}\left(\left(\sqrt{m + d\log(1+\Delta_T)} + \sqrt{\Delta_T}\right) \cdot \sqrt{mT}\right)$	$O(nmdT)$	$O(nmd)$
Random Projections			
CBRAP [19]	$\tilde{O}\left(k\sqrt{T} + \varepsilon\sqrt{k}T\right)$	$O(nk^2T)$	$O(nk^2)$
ConfidenceBall1_FJLT (**section III-B**)	$\tilde{O}\left(k\sqrt{T} + 2\varepsilon T\right)$	$O(nk^2T)$	$O(nk^2)$

Evaluation settings: We consider two settings:

(i) **Contextual** (Figure 1). At each iteration a one-hot user context and its itemset are sampled randomly, and the reward when recommending an item has is binary: $r_t = 1$ if the user has interacted with the item and 0 otherwise. Consequently, the cumulative reward averaged over time is equivalent to the click-through rate (CTR). A higher CTR indicates better recommendations.

(ii) **Non-contextual** (Figures 2—5 and Tables III—IV). A fixed user and action set are selected at the beginning, and the reward $r_t = \langle \theta_*, a_t \rangle + \eta$, where $\eta \sim \mathcal{N}(0, 0.1^2)$. The performance in this setting is evaluated via cumulative regret. A lower cumulative regret signifies better alignment with user preferences.

Implementation details: Results are obtained by running each algorithm 20–50 times (with fixed seeds) Hyperparameters $\lambda \in \{2 \times 10^{-11}, \ldots, 2 \times 10^{2}\}$, $scale \in \{10^{-6}, \ldots, 10^{1}\}$, sketch size m (CBSCFD, SOFUL) or projection size k (ConfidenceBall1-FJLT, CBRAP) are optimized via grid search, retaining the best configuration.

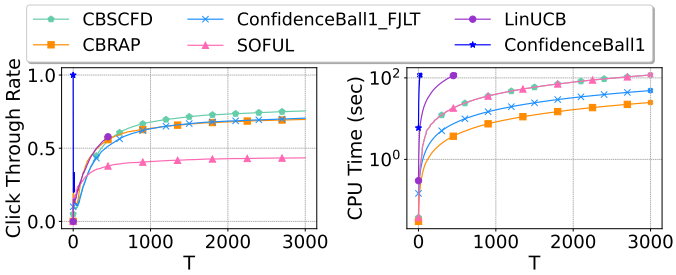
We conduct all experiments on a Linux server having an 8-core bi-pro Intel Xeon E5-2609v2 2.5GHz processor and total memory of 64GB. The open-source Python implementation is available online.[6] The raw experimental results are also available online.[7]

B. Experimental Results

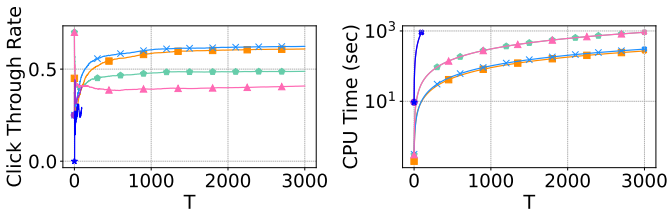
In this section, we present the results obtained for various datasets and algorithms tested. For each plot and for each algorithm, we report the best outcomes achieved in terms of cumulative regret or CTR averaged across the different combinations of parameters tested ($scale$ and λ). We selected the results that we consider the most relevant; additional results on other datasets and supplementary information are available online. [6]

[7]https://www.lri.fr/~izri/results.zip

Click-Through Rate: Figures 1a and 1b show the CTR and CPU time for the different datasets and algorithms. In Figure 1a, the best-performing algorithm in terms of CTR is CBSCFD. The latter seems to achieve the same result as the baseline algorithm LINUCB, which, along with CONFIDENCEBALL1, was stopped early due to excessive processing time. Figure 1b shows that algorithms using random projections, namely CBRAP and CONFIDENCEBALL1-FJLT, perform better on the Steam dataset than the ones using sketches (i.e. CBSCFD, SOFUL). On the other hand, all results show that SOFUL achieves the lowest CTR across all data sets.



(a) Yahoo



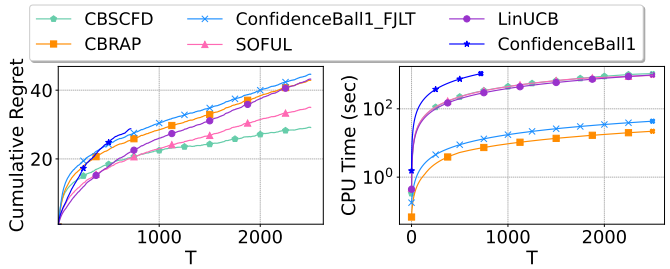
(b) Steam

Fig. 1: Click-through rate and CPU time per dataset. Contextual setting.

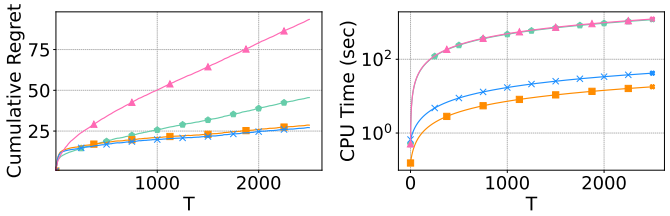
Cumulative Regret: Figures 2a, 2b and Table III report the cumulative regret and CPU time for different datasets and algorithms. In Table III, for each metric, the best algorithm is in bold and the second-best is underlined. In general, the best-performing algorithms in terms of average cumulative regret

are CBSCFD and SOFUL (i.e., the algorithms that sample the matrix). These algorithms achieve results similar or better than baseline algorithms, as can be seen for example in Figure 2a. Conversely, algorithms using random projections, namely CBRAP and CONFIDENCEBALL1-FJLT are less efficient. There are a few exceptions, notably in the case of the Steam dataset (Figure 2b), and even more so in synthetic data (Table III), where CBRAP and CONFIDENCEBALL1-FJLT perform better, while other algorithms suffer from a higher regret which seems linear.

Variations in whether sketch based algorithms outperform random projection based algorithms (for instance Figure 2a where they do in Figure 2b where they do not) are due to combining results from different runs and where users having different reward distribution are selected for each run; this is the case in the MovieLens, Steam, Yahoo and Amazon datasets. Moreover, the parameters set cannot be optimal for all users and algorithms, resulting in sub-linear regret for some users and linear regret for others. Moreover, as discussed in Section III, sketching-based algorithms can provide more accurate approximations than random projection-based methods, leading to lower regret. However, they may still suffer linear regret in certain scenarios.



(a) MovieLens



(b) Steam

Fig. 2: Regret and CPU time comparison per dataset. Non-Contextual Setting.

CPU Time: In contrast to regret, the fastest algorithms are always the ones that use random projections. In the case of dataset Synthetic 1 ($d = 100$ and $n = 200$) for example, we can see on Table III that CBRAP is 5 times faster than CBSCFD and SOFUL. In the case of datasets where the initial dimension and the size of the itemset are much larger (e.g. steam with $d = 1803$, $n = 2000$), we can see a more significant gap in the CPU time between two types of algorithms. Indeed, as we can see on the right of Figure 2b, CONFIDENCEBALL1-FJLT is up to 30 times faster than CBSCFD and SOFUL in terms of CPU time, while CBRAP is 60 times faster than those algorithms. We cannot apply the baseline algorithms on large datasets, as they exceed our time limit (set to the maximum runtime of other algorithms), even within 500 iterations.

TABLE III: Regret and CPU Time for Synthetic Data at T=2000.

	Synthetic 1 $d = 100,\ n = 200$		**Synthetic 2** $d = 1000,\ n = 2000$	
Model	**Regret**	**CPU Time**	**Regret**	**CPU Time**
CBall1	74.16	50s	-	-
LinUCB	63.65	24s	-	-
CBSCFD	55.23	23s	531.77	490s
SOFUL	651.97	21s	648.67	470s
CBRAP	**42.29**	**4s**	311.00	**15s**
CBall1_FJLT	73.94	20s	**207.90**	34s

The difference between algorithms that use random projections and the ones that sketch the covariance matrix is due to the fact that algorithms using sketches have a time complexity that always depends on the initial dimension d, whereas algorithms using projection matrices have a time complexity that only depends on the new reduced dimension.

Memory Usage: As shown in Figure 3, sketch-based algorithms require significantly more memory than random projection methods on high-dimensional datasets (e.g., MovieLens), especially when the sketch size must be large enough to ensure low regret. This comes from their $O(nd)$ per iteration cost due to matrix multiplications and storage of S and H (of sizes $O(md)$ and $O(m^2)$), in addition to the item set and parameter estimator. In contrast, random projection methods store only the covariance matrix $O(k^2)$ and projected itemset $O(nk)$, with UCB computations costing $O(nk)$.

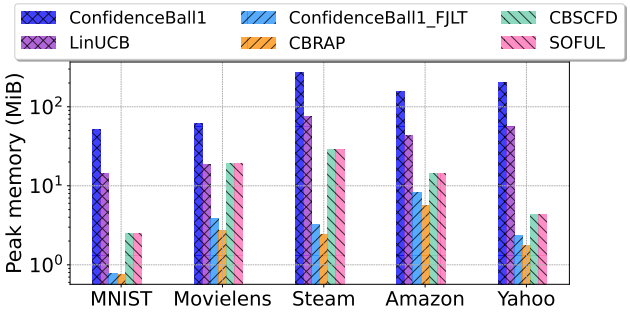


Fig. 3: Peak of memory allocation per dataset.

Parameter Sensitivity Analysis: Figure 4 displays the final (cumulative) regret values in function of the regularization parameter λ and the confidence bound scale. Each algorithm has an optimal value that allows us to get the best cumulative regret; importantly, the algorithms' variance is the lowest at this specific value. Moreover, the optimal value for the confidence bound is not the one that is found theoretically; evidence points that scale factors closer to 0.1 or 0.01 are more optimal. Naturally, once we decrease the scale too much (thus practically removing the impact of the confidence bound), the results are worse generally. The difficulty is that bandits, being

online algorithms, are not amenable to hyper-parameter tuning; nevertheless, the results show that values around 0.01 for scale and 10^{-5} for λ generate better results. Similar findings have been observed on other datasets.

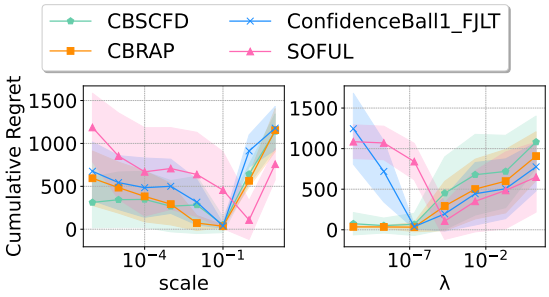


Fig. 4: Impact of parameters on regret on Steam dataset.

Impact of Sketch and Projection Dimension on Regret: Experiments on the MovieLens dataset show that, when using random projections, increasing the dimensionality of the projection result in more sublinear regret (in the sense that regret growth decays more visibly), as shown in Figure 5a and 5b. This is unsurprising: the projected dimension gets closer to the initial dimension and thus the theoretical guarantees of the baseline model. However, increasing the sketch size in sketch-based algorithms does not always improve the results. For example, the results of CBSCFD algorithm in Figure 5c suggest that increasing the sketch size initially leads to lower sub-linear regret with larger sketches, before reverting to earlier behavior. For SOFUL, in Figure 5d, increasing the sketch size does not appear to improve the results.

This experiment shows that increasing the values of k or m not only increases memory usage and computation time, but also leads to an increase in the average cumulative regret. This indicates that asymptotic behavior may not always be the best criterion for the horizons we considered here. More specifically, achieving sub-linear regret should not always be the end goal in practice: one should prefer steadily increasing linear regret that is lower at the end of the iterations than a quickly increasing sub-linear regret shape. Therefore, increasing the size of the sketch or the projected dimension has an impact beyond resource consumption and running time, by potentially negatively affecting the minimization of cumulative regret.

`for` loop vs. Matrix Multiplication: All results presented in this paper have been obtained using matrix multiplications only. Another possibility is to implement the presented algorithms more directly. Indeed, instead of performing a matrix multiplication involving the item set (all items at once), one can use a `for` loop to iterate through the list of items and estimate the gain of each individually.

In Figure 6, we compare the use of `for` loops implementation to matrix multiplications one, in terms of CPU time and memory usage for the Movielens dataset in the contextual setting and having $m = 10$, $k = 50$ and $T = 200$. As we can see the use of matrix multiplications leads to faster algorithms in terms of time, as shown in Figure 6, particularly for the CBRAP algorithm. However, we can also notice that this time efficiency leads to much higher memory consumption, especially for algorithms using sketches, as intermediate computations require the storage of matrices of order $n \times d$.

Consequently, the use of `for` loops appears to be a more memory-efficient approach, facilitating the manipulation of larger datasets on devices having low memory. Conversely, using matrix multiplications consumes more memory, especially if the data set is large (high dimension d and large number of items n), although this may be less of a concern if the memory is more plentiful.

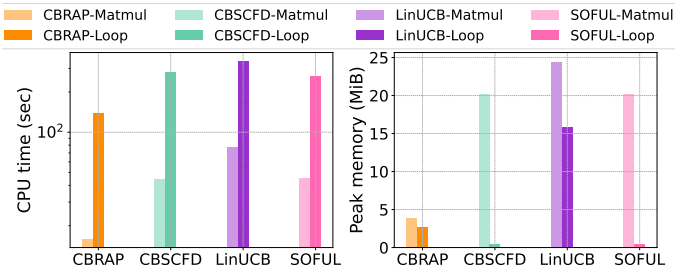


Fig. 6: CPU time and Memory usage comparison per implementation for Movielens dataset.

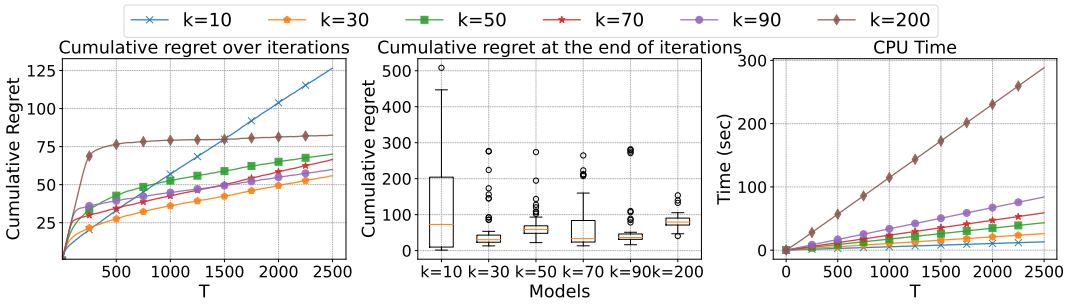
VI. Discussion and Conclusions

The results presented in the experimental study show that the choice of the type of approach for space- and time-savings for recommendation using linear bandits is not entirely straightforward and depends greatly on the dataset. To aid practitioners, we conclude by discussing these trade-offs and giving some takeaways and insights for practical use.

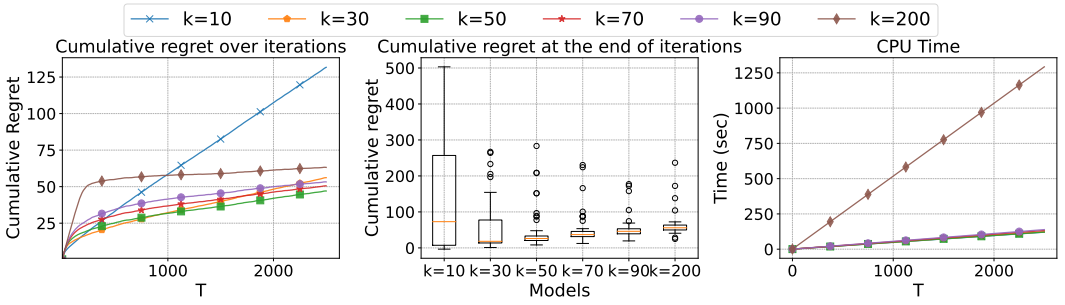
Table IV presents the trade-off between regret, CPU time, and memory per dataset. The algorithms which are Pareto efficient i.e., not dominated by any other, appear in bold in the table.

TABLE IV: Cumulative Regret (R), Time in seconds (T), and Memory in MiB (M) for each model and dataset.

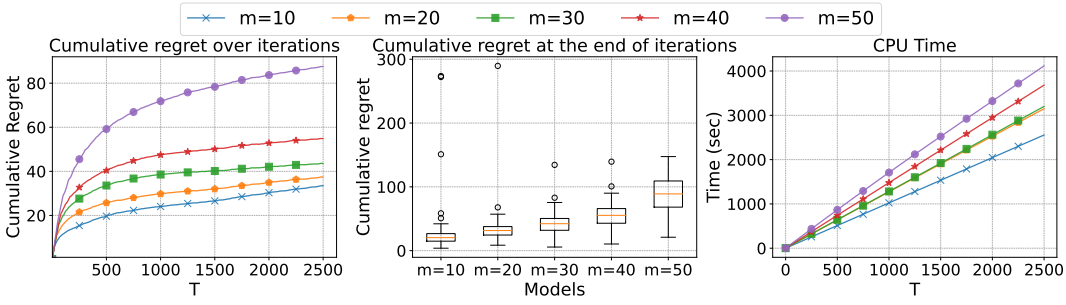
Dataset	Metric	CBall1	LinUCB	CBSCFD	SOFUL	CBRAP	CBall1_FJLT
MNIST	R	328	**311**	**317**	443	**597**	**573**
	T	2850	**131**	**41**	42	**15**	**35**
	M	51	**14**	**2.5**	2.5	**0.7**	**0.7**
MovieLens	R	39	**43**	**33**	**34**	**47**	69
	T	3749	**1078**	**1071**	**950**	**22**	43
	M	61	**18**	**19**	**19**	**2.7**	3.7
Steam	R	—	—	52	107	**35**	**38**
	T	—	—	1183	1232	**18**	**42**
	M	—	—	28	28	**2.4**	**3.2**
Yahoo	R	122	98	**100**	612	**103**	123
	T	13367	680	**115**	123	**10**	59
	M	203	55	**4**	4	**1.7**	2.3
Amazon	R	260	**190**	339	469	**282**	282
	T	10607	**1203**	533	530	**36**	241
	M	156	**42**	14	14	**5.6**	8



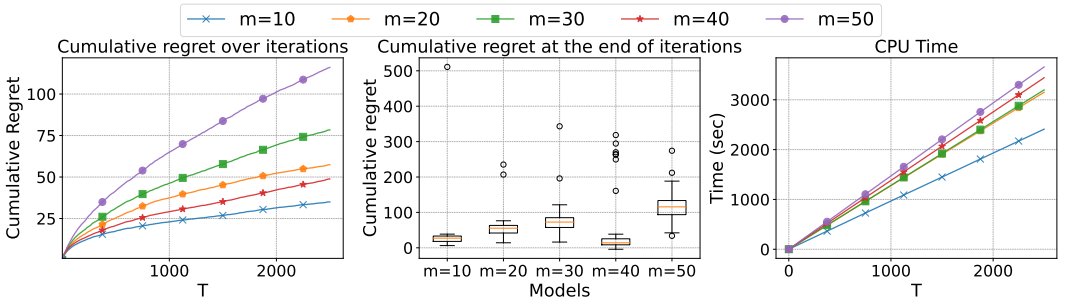
(a) ConfidenceBall1_FJLT



(b) CBRAP



(c) CBSCFD



(d) SOFUL

Fig. 5: Impact of sketch and projection dimension on regret.

This table shows that, when one prioritizes fast execution, the approaches based on projecting the data matrix (ConfidenceBall1-FJLT and CBRAP) achieve the lowest running time by a significant margin. In some datasets, this is achieved with a modest increase in regret, despite the weaker theoretical guarantees compared to the sketching-based approaches. However, they can exhibit linear regret if the projection dimension is insufficiently representative of the data. In contrast, sketching algorithms, CBSCFD and SOFUL, typically achieve lower regret, although they are more time consuming (and sometimes even as costly as the baseline algorithms, LinUCB and ConfidenceBall1, see MovieLens).

The table also shows that approaches based on random projections are also more memory-efficient than the sketching algorithms (SOFUL and CBSCFD), especially when the dimension of action vectors d is high as in the case of the Steam dataset. The lower efficiency of sketching algorithms reflects their theoretical dependency on d.

Our experimental study emphasizes that the choice of

method is strongly influenced by the specific requirements of the application. Random projections are preferable for applications that prioritize speed or memory, and which tolerate a slightly larger regret, while sketching algorithms are preferable when minimizing regret is crucial, despite higher computational and memory costs, as they keep the regret within reasonable bounds both theoretically and in practice.

We believe that these results show that simple algorithms such as linear multi-armed bandits are useful for practitioners, as they can be augmented with space- and time-saving techniques to achieve good results in practical application domains, as evidenced by our recommendation scenario.

ACKNOWLEDGMENTS

This work is partially supported by by the National Research Foundation, Prime Minister's Office, Singapore under its Campus for Research Excellence and Technological Enterprise (CREATE), project DesCartes, and by the French funding agency ANR project France 2030 ANR-23-IACL-0006.

REFERENCES

[1] H. Robbins, "Some aspects of the sequential design of experiments," *Bulletin of the American Mathematical Society*, vol. 58, no. 5, pp. 527–535, 1952. [Online]. Available: https://www.ams.org/bull/1952-58-05/S0002-9904-1952-09620-8/

[2] M. Fujisawa, H. Yasuda, R. Isogai, M. Arai, Y. Yoshida, A. Li, S.-J. Kim, and M. Hasegawa, "An efficient beaconing of bluetooth low energy by decision making algorithm," *Discover Artificial Intelligence*, vol. 4, 04 2024.

[3] A. Musaddiq, T. Olsson, and F. Ahlgren, "Reinforcement-learning-based routing and resource management for internet of things environments: Theoretical perspective and challenges," *Sensors*, vol. 23, p. 8263, 10 2023.

[4] P. Auer, "Using confidence bounds for exploitation-exploration tradeoffs," *The Journal of Machine Learning Research*, vol. 3, no. null, pp. 397–422, Mar. 2003.

[5] P. Auer, N. Cesa-Bianchi, and P. Fischer, "Finite-time analysis of the multiarmed bandit problem," *Machine learning*, vol. 47, pp. 235–256, 2002.

[6] L. Wei and V. Srivastava, "Nonstationary stochastic bandits: Ucb policies and minimax regret," *IEEE Open Journal of Control Systems*, vol. PP, pp. 1–14, 01 2024.

[7] Y. Abbasi-yadkori, D. Pál, and C. Szepesvári, "Improved Algorithms for Linear Stochastic Bandits," in *Advances in Neural Information Processing Systems*, vol. 24. Curran Associates, Inc., 2011.

[8] D. Bouneffouf, I. Rish, and C. C. Aggarwal, "Survey on applications of multi-armed and contextual bandits," in *IEEE Congress on Evolutionary Computation, CEC 2020, Glasgow, United Kingdom, July 19-24, 2020*. IEEE, 2020, pp. 1–8. [Online]. Available: https://doi.org/10.1109/CEC48606.2020.9185782

[9] L. Li, W. Chu, J. Langford, and R. E. Schapire, "A contextual-bandit approach to personalized news article recommendation," *CoRR*, vol. abs/1003.0146, 2010. [Online]. Available: http://arxiv.org/abs/1003.0146

[10] Q. Ding, C. Hsieh, and J. Sharpnack, "An efficient algorithm for generalized linear bandit: Online stochastic gradient descent and thompson sampling," *CoRR*, vol. abs/2006.04012, 2020. [Online]. Available: https://arxiv.org/abs/2006.04012

[11] S. Gupta, S. Chaudhari, G. Joshi, and O. Yagan, "Multi-armed bandits with correlated arms," *IEEE Transactions on Information Theory*, vol. 67, no. 10, p. 6711–6732, Oct. 2021. [Online]. Available: http://dx.doi.org/10.1109/TIT.2021.3081508

[12] V. Dani, T. P. Hayes, and S. M. Kakade, "Stochastic linear optimization under bandit feedback," in *21st Annual Conference on Learning Theory*, no. 101, 2008, pp. 355–366.

[13] W. Chu, L. Li, L. Reyzin, and R. Schapire, "Contextual Bandits with Linear Payoff Functions," in *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*. JMLR Workshop and Conference Proceedings, Jun. 2011, pp. 208–214, iSSN: 1938-7228. [Online]. Available: https://proceedings.mlr.press/v15/chu11a.html

[14] T. Lattimore and C. Szepesvári, *Bandit Algorithms*, 1st ed. Cambridge University Press, Jul. 2020. [Online]. Available: https://www.cambridge.org/core/product/identifier/9781108571401/type/book

[15] M. Ghashami, E. Liberty, J. M. Phillips, and D. P. Woodruff, "Frequent directions: Simple and deterministic matrix sketching," *SIAM Journal on Computing*, vol. 45, no. 5, pp. 1762–1792, 2016.

[16] I. Kuzborskij, L. Cella, and N. Cesa-Bianchi, "Efficient Linear Bandits through Matrix Sketching," in *Proceedings of the Twenty-Second International Conference on Artificial Intelligence and Statistics*. PMLR, Apr. 2019, pp. 177–185, iSSN: 2640-3498. [Online]. Available: https://proceedings.mlr.press/v89/kuzborskij19a.html

[17] C. Chen, L. Luo, W. Zhang, Y. Yu, and Y. Lian, "Efficient and Robust High-Dimensional Linear Contextual Bandits," in *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence*. Yokohama, Japan: International Joint Conferences on Artificial Intelligence Organization, Jul. 2020, pp. 4259–4265. [Online]. Available: https://www.ijcai.org/proceedings/2020/588

[18] W. B. Johnson and J. Lindenstrauss, "Extensions of Lipschitz mappings into a Hilbert space," in *Contemporary Mathematics*, R. Beals, A. Beck, A. Bellow, and A. Hajian, Eds. Providence, Rhode Island: American Mathematical Society, 1984, vol. 26, pp. 189–206. [Online]. Available: http://www.ams.org/conm/026/

[19] X. Yu, M. R. Lyu, and I. King, "CBRAP: Contextual Bandits with RAndom Projection," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 31, no. 1, Feb. 2017, number: 1. [Online]. Available: https://ojs.aaai.org/index.php/AAAI/article/view/10888

[20] N. Ailon and B. Chazelle, "The Fast Johnson–Lindenstrauss Transform and Approximate Nearest Neighbors," *SIAM Journal on Computing*, vol. 39, no. 1, pp. 302–322, Jan. 2009. [Online]. Available: http://epubs.siam.org/doi/10.1137/060673096

[21] X. Wang, M. M. Wei, and T. Yao, "Efficient Sparse Linear Bandits under High Dimensional Data," in *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*. Long Beach CA USA: ACM, Aug. 2023, pp. 2431–2443. [Online]. Available: https://dl.acm.org/doi/10.1145/3580305.3599329

[22] S. Lale, K. Azizzadenesheli, A. Anandkumar, and B. Hassibi, "Stochastic linear bandits with hidden low rank structure," *arXiv preprint arXiv:1901.09490*, 2019.

[23] D. Calandriello, L. Carratino, A. Lazaric, M. Valko, and L. Rosasco, "Gaussian Process Optimization with Adaptive Sketching: Scalable and No Regret," in *Proceedings of the Thirty-Second Conference on Learning Theory*. PMLR, Jun. 2019, pp. 533–557, iSSN: 2640-3498. [Online]. Available: https://proceedings.mlr.press/v99/calandriello19a.html

[24] H. Zenati, A. Bietti, E. Diemert, J. Mairal, M. Martin, and P. Gaillard, "Efficient Kernelized UCB for Contextual Bandits," in *Proceedings of The 25th International Conference on Artificial Intelligence and Statistics*. PMLR, May 2022, pp. 5689–5720, iSSN: 2640-3498. [Online]. Available: https://proceedings.mlr.press/v151/zenati22a.html

[25] M. Valko, N. Korda, R. Munos, I. Flaounas, and N. Cristianini, "Finite-Time Analysis of Kernelised Contextual Bandits," Sep. 2013, arXiv:1309.6869 [cs, stat]. [Online]. Available: http://arxiv.org/abs/1309.6869

[26] S. Ghoorchian, E. Kortukov, and S. Maghsudi, "Non-Stationary Linear Bandits With Dimensionality Reduction for Large-Scale Recommender Systems," *IEEE Open Journal of Signal Processing*, vol. PP, pp. 1–12, Jan. 2024.

[27] S. Agrawal and N. Goyal, "Thompson sampling for contextual bandits with linear payoffs," in *Proceedings of the 30th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, S. Dasgupta and D. McAllester, Eds., vol. 28, no. 3. Atlanta, Georgia, USA: PMLR, 17–19 Jun 2013, pp. 127–135. [Online]. Available: https://proceedings.mlr.press/v28/agrawal13.html