Contextual Bandits with Large Action Spaces: Made Practical

Yinglun Zhu [1] **Dylan Foster** [2] **John Langford** [2] **Paul Mineiro** [2]

Abstract

A central problem in sequential decision making is to develop algorithms that are practical and computationally efficient, yet support the use of flexible, general-purpose models. Focusing on the contextual bandit problem, recent progress provides provably efficient algorithms with strong empirical performance when the number of possible alternatives ("actions") is small, but guarantees for decision making in large, continuous action spaces have remained elusive, leading to a significant gap between theory and practice. We present the first efficient, general-purpose algorithm for contextual bandits with continuous, linearly structured action spaces. Our algorithm makes use of computational oracles for (i) supervised learning, and (ii) optimization over the action space, and achieves sample complexity, runtime, and memory independent of the size of the action space. In addition, it is simple and practical. We perform a large-scale empirical evaluation, and show that our approach typically enjoys superior performance and efficiency compared to standard baselines.

1. Introduction

We consider the design of practical, theoretically motivated algorithms for sequential decision making with contextual information, better known as the *contextual bandit problem*. Here, a learning agent repeatedly receives a *context* (e.g., a user's profile), selects an *action* (e.g., a news article to display), and receives a *reward* (e.g., whether the article was clicked). Contextual bandits are a useful model for decision making in unknown environments in which both exploration and generalization are required, but pose significant algorithm design challenges beyond classical supervised learning. Recent years have seen development on two fronts: On the theoretical side, extensive research into finite-action contextual bandits has resulted in practical, provably efficient algorithms capable of supporting flexible, general-purpose models (Langford & Zhang, 2007; Agarwal et al., 2014; Foster & Rakhlin, 2020; Simchi-Levi & Xu, 2021; Foster & Krishnamurthy, 2021). Empirically, contextual bandits have been widely deployed in practice for online personalization and recommendation tasks (Li et al., 2010; Agarwal et al., 2016; Tewari & Murphy, 2017; Cai et al., 2021), leveraging the availability of high-quality action slates (e.g., subsets of candidate articles selected by an editor).

Both developments above critically rely on the existence of a small number of possible decisions or alternatives. However, many applications demand the ability to make contextual decisions in large, potentially continuous spaces, where actions might correspond to images in a database or high-dimensional embeddings of rich documents such as webpages. Contextual bandits in large (e.g., million-action) settings remains a major challenge—both statistically and computationally—and constitutes a substantial gap between theory and practice. In particular:

- Existing *general-purpose* algorithms (Langford & Zhang, 2007; Agarwal et al., 2014; Foster & Rakhlin, 2020; Simchi-Levi & Xu, 2021; Foster & Krishnamurthy, 2021) allow for the use of flexible models (e.g., neural networks, forests, or kernels) to facilitate generalization across contexts, but have sample complexity and computational requirements linear in the number of actions. These approaches can degrade in performance under benign operations such as duplicating actions.
- While certain recent approaches extend the general-purpose methods above to accommodate large action spaces, they either require sample complexity exponential in action dimension (Krishnamurthy et al., 2020), or require additional distributional assumptions (Sen et al., 2021).
- Various results efficiently handle large or continuous action spaces (Dani et al., 2008; Jun et al., 2017; Yang et al., 2021) with specific types of function approximation, but do not accommodate general-purpose models.

[1]University of Wisconsin-Madison [2]Microsoft Research NYC. Correspondence to: Yinglun Zhu <yinglun@cs.wisc.edu>.

Proceedings of the 39^{th} International Conference on Machine Learning, Baltimore, Maryland, USA, PMLR 162, 2022.

As a result of these algorithmic limitations, empirical aspects of contextual decision making in large action spaces have remained relatively unexplored compared to the small-action regime (Bietti et al., 2021), with little in the way of readily deployable out-of-the-box solutions.

Contributions. We provide the first efficient algorithms for contextual bandits with continuous, linearly structured action spaces and general function approximation. Following Chernozhukov et al. (2019); Xu & Zeevi (2020); Foster et al. (2020), we adopt a modeling approach, and assume rewards for each context-action pair (x, a) are structured as

$$f^\star(x, a) = \langle \phi(x, a), g^\star(x) \rangle. \tag{1}$$

Here $\phi(x, a) \in \mathbb{R}^d$ is a known context-action embedding (or feature map) and $g^\star \in \mathcal{G}$ is a context embedding to be learned online, which belongs to an arbitrary, user-specified function class $\mathcal{G}$. Our algorithm, SpannerIGW, is computationally efficient (in particular, the runtime and memory are *independent* of the number of actions) whenever the user has access to (i) an *online regression oracle* for supervised learning over the reward function class, and (ii) an *action optimization oracle* capable of solving problems of the form

$$\arg\max_{a \in \mathcal{A}} \langle \phi(x, a), \theta \rangle$$

for any $\theta \in \mathbb{R}^d$. The former oracle follows prior approaches to finite-action contextual bandits (Foster & Rakhlin, 2020; Simchi-Levi & Xu, 2021; Foster & Krishnamurthy, 2021), while the latter generalizes efficient approaches to (non-contextual) linear bandits (McMahan & Blum, 2004; Dani et al., 2008; Bubeck et al., 2012; Hazan & Karnin, 2016). We provide a regret bound for SpannerIGW which scales as $\sqrt{\mathrm{poly}(d) \cdot T}$, and—like the computational complexity—is independent of the number of actions. Beyond these results, we provide a particularly practical variant of SpannerIGW (SpannerGreedy), which enjoys even faster runtime at the cost of slightly worse ($\mathrm{poly}(d) \cdot T^{2/3}$-type) regret.

Our techniques. On the technical side, we show how to *efficiently* combine the inverse gap weighting technique (Abe & Long, 1999; Foster & Rakhlin, 2020) previously used in the finite-action setting with optimal design-based approaches for exploration with linearly structured actions. This offers a computational improvement upon the results of Xu & Zeevi (2020); Foster et al. (2020), which provide algorithms with $\sqrt{\mathrm{poly}(d) \cdot T}$-regret for the setting we consider, but require enumeration over the action space. Conceptually, our results expand upon the class of problems for which minimax approaches to exploration (Foster et al., 2021b) can be made efficient.

Empirical performance. As with previous approaches based on regression oracles, SpannerIGW is simple, practical, and well-suited to flexible, general-purpose function approximation. In extensive experiments ranging from thousands to millions of actions, we find that our methods typically enjoy superior performance compared to existing baselines. In addition, our experiments validate the statistical model in Eq. (1) which we find to be well-suited to learning with large-scale language models (Devlin et al., 2019).

1.1. Organization

This paper is organized as follows. In Section 2, we formally introduce our statistical model and the computational oracles upon which our algorithms are built. Subsequent sections are dedicated to our main results.

- As a warm-up, Section 3 presents a simplified algorithm, SpannerGreedy, which illustrates the principle of exploration over an approximate optimal design. This algorithm is practical and oracle-efficient, but has suboptimal $\mathrm{poly}(d) \cdot T^{2/3}$-type regret.
- Building on these ideas, Section 4 presents our main algorithm, SpannerIGW, which combines the idea of approximate optimal design used by SpannerGreedy with the inverse gap weighting method (Abe & Long, 1999; Foster & Rakhlin, 2020), resulting in an oracle-efficient algorithm with $\sqrt{\mathrm{poly}(d) \cdot T}$-regret.

Section 5 presents empirical results for both algorithms. We close with a discussion of future directions (Section 6). Additional related work and proof are deferred to the appendix.

2. Problem Setting

The contextual bandit problem proceeds over T rounds. At each round $t \in [T]$, the learner receives a context $x_t \in \mathcal{X}$ (the *context space*), selects an action $a_t \in \mathcal{A}$ (the *action space*), and then observes a reward $r_t(a_t)$, where $r_t : \mathcal{A} \to [-1, 1]$ is the underlying reward function. We assume that for each round t, conditioned on x_t, the reward r_t is sampled from a (unknown) distribution $\mathbb{P}_{r_t}(\cdot \mid x_t)$. We allow both the contexts $x_1, \ldots, x_T$ and the distributions $\mathbb{P}_{r_1}, \ldots, \mathbb{P}_{r_T}$ to be selected in an arbitrary, potentially adaptive fashion based on the history.

Function approximation. Following a standard approach to developing efficient contextual bandit methods, we take a modeling approach, and work with a user-specified class of regression functions $\mathcal{F} \subseteq (\mathcal{X} \times \mathcal{A} \to [-1, 1])$ that aims to model the underlying mean reward function. We make the following realizability assumption (Agarwal et al., 2012; Foster et al., 2018; Foster & Rakhlin, 2020; Simchi-Levi & Xu, 2021).

Assumption 1 (Realizability). *There exists a regression function $f^\star \in \mathcal{F}$ such that $\mathbb{E}[r_t(a) \mid x_t = x] = f^\star(x, a)$ for all $a \in \mathcal{A}$ and $t \in [T]$.*

Without further assumptions, there exist function classes $\mathcal{F}$ for which the regret of any algorithm must grow proportionally to $|\mathcal{A}|$ (e.g., Agarwal et al. (2012)). In order to facilitate generalization across actions and achieve sample complexity and computational complexity independent of $|\mathcal{A}|$, we assume that each function $f \in \mathcal{F}$ is linear in a known (context-dependent) feature embedding of the action. Following Xu & Zeevi (2020); Foster et al. (2020), we assume that $\mathcal{F}$ takes the form

$$\mathcal{F} = \{f_g(x,a) = \langle \phi(x,a), g(x) \rangle : g \in \mathcal{G}\},$$

where $\phi(x,a) \in \mathbb{R}^d$ is a known, context-dependent action embedding and $\mathcal{G}$ is a user-specified class of context embedding functions.

This formulation assumes linearity in the action space (after featurization), but allows for nonlinear, *learned* dependence on the context x through the function class $\mathcal{G}$, which can be taken to consist of neural networks, forests, or any other flexible function class a user chooses. For example, in news article recommendation, $\phi(x,a) = \phi(a)$ might correspond to an embedding of an article a obtained using a large pre-trained language-model, while $g(x)$ might correspond to a task-dependent embedding of a user x, which our methods can learn online. Well-studied special cases include the linear contextual bandit setting (Chu et al., 2011; Abbasi-Yadkori et al., 2011), which corresponds to the special case where each $g \in \mathcal{G}$ has the form $g(x) = \theta$ for some fixed $\theta \in \mathbb{R}^d$, as well as the standard finite-action contextual bandit setting, where $d = |\mathcal{A}|$ and $\phi(x,a) = e_a$.

We let $g^\star \in \mathcal{G}$ be the embedding for which $f^\star = f_{g^\star}$. We assume for simplicity that $\sup_{x\in\mathcal{X},a\in\mathcal{A}} \|\phi(x,a)\| \leq 1$ and $\sup_{g\in\mathcal{G},x\in\mathcal{X}} \|g(x)\| \leq 1$. In addition, we assume that $\mathrm{span}(\{\phi(x,a)\}) = \mathbb{R}^d$ for all $x \in \mathcal{X}$.

Regret. For each regression function $f \in \mathcal{F}$, let $\pi_f(x_t) := \arg\max_{a\in\mathcal{A}} f(x_t,a)$ denote the induced policy, and define $\pi^\star := \pi_{f^\star}$ as the optimal policy. We measure the performance of the learner in terms of regret to $\pi^\star$:

$$\mathbf{Reg}_{\mathsf{CB}}(T) := \sum_{t=1}^{T} r_t(\pi^\star(x_t)) - r_t(a_t).$$

2.1. Computational Oracles

To derive efficient algorithms with sublinear runtime, we make use of two computational oracles: First, following Foster & Rakhlin (2020); Simchi-Levi & Xu (2021); Foster et al. (2020; 2021a), we use an *online regression oracle* for supervised learning over the reward function class $\mathcal{F}$. Second, we use an *action optimization oracle*, which facilitates linear optimization over the action space $\mathcal{A}$ (McMahan & Blum, 2004; Dani et al., 2008; Bubeck et al., 2012; Hazan & Karnin, 2016)

Function approximation: Regression oracles. A fruitful approach to designing efficient contextual bandit algorithms is through reduction to supervised regression with the class $\mathcal{F}$, which facilitates the use of off-the-shelf supervised learning algorithms and models (Foster & Rakhlin, 2020; Simchi-Levi & Xu, 2021; Foster et al., 2020; 2021a). Following Foster & Rakhlin (2020), we assume access to an *online regression oracle* $\mathbf{Alg}_{\mathsf{Sq}}$, which is an algorithm for online learning (or, sequential prediction) with the square loss.

We consider the following protocol. At each round $t \in [T]$, the oracle produces an estimator $\widehat{f}_t = f_{\widehat{g}_t}$, then receives a context-action-reward tuple $(x_t, a_t, r_t(a_t))$. The goal of the oracle is to accurately predict the reward as a function of the context and action, and we evaluate its performance via the square loss $(\widehat{f}_t(x_t,a_t) - r_t)^2$. We measure the oracle's cumulative performance through square-loss regret to $\mathcal{F}$, defined below.

Assumption 2 (Bounded square-loss regret)**.** *The regression oracle $\mathbf{Alg}_{\mathsf{Sq}}$ guarantees that for any (potentially adaptively chosen) sequence $\{(x_t, a_t, r_t(a_t))\}_{t=1}^T$,*

$$\sum_{t=1}^{T}\Big(\widehat{f}_t(x_t,a_t) - r_t(a_t)\Big)^2 - \inf_{f\in\mathcal{F}} \sum_{t=1}^{T} (f(x_t,a_t) - r_t(a_t))^2 \leq \mathbf{Reg}_{\mathsf{Sq}}(T),$$

for some (non-data-dependent) function $\mathbf{Reg}_{\mathsf{Sq}}(T)$.

We let $\mathcal{T}_{\mathsf{Sq}}$ denote an upper bound on the time required to (i) query the oracle's estimator $\widehat{g}_t$ with x_t and receive the vector $\widehat{g}_t(x_t) \in \mathbb{R}^d$, and (ii) update the oracle with the example $(x_t, a_t, r_t(a_t))$. We let $\mathcal{M}_{\mathsf{Sq}}$ denote the maximum memory used by the oracle throughout its execution.

Online regression is a well-studied problem, with computationally efficient algorithms for many models. Basic examples include finite classes $\mathcal{F}$, where one can attain $\mathbf{Reg}_{\mathsf{Sq}}(T) = O(\log|\mathcal{F}|)$ (Vovk, 1998), and linear models ($g(x) = \theta$), where the online Newton step algorithm (Hazan et al., 2007) satisfies Assumption 2 with $\mathbf{Reg}_{\mathsf{Sq}}(T) = O(d \log T)$. More generally, even for classes such as deep neural networks for which provable guarantees may not be available, regression is well-suited to gradient-based methods. We refer to Foster & Rakhlin (2020); Foster et al. (2020) for more comprehensive discussion.

Large action spaces: Action optimization oracles. The regression oracle setup in the prequel is identical to that considered in the finite-action setting (Foster & Rakhlin, 2020). In order to develop efficient algorithms for large or infinite action spaces, we assume access to an oracle for linear optimization over actions.

Definition 1 (Action optimization oracle)**.** *An* action optimization oracle $\mathbf{Alg}_{\mathsf{Opt}}$ *takes as input a context $x \in \mathcal{X}$, and*

vector $\theta \in \mathbb{R}^d$ and returns

$$a^\star := \arg\max_{a \in \mathcal{A}} \langle \phi(x, a), \theta \rangle. \tag{2}$$

For a single query to the oracle, We let $\mathcal{T}_{\mathsf{Opt}}$ denote a bound on the runtime for a single query to the oracle. We let $\mathcal{M}_{\mathsf{Opt}}$ denote the maximum memory used by the oracle throughout its execution.

The action optimization oracle in Eq. (2) is widely used throughout the literature on linear bandits (Dani et al., 2008; Chen et al., 2017; Cao & Krishnamurthy, 2019; Katz-Samuels et al., 2020), and can be implemented in polynomial time for standard combinatorial action spaces. It is a basic computational primitive in the theory of convex optimization, and when $\mathcal{A}$ is convex, it is equivalent (up to polynomial-time reductions) to other standard primitives such as separation oracles and membership oracles (Schrijver, 1998; Grötschel et al., 2012). It also equivalent to the well-known Maximum Inner Product Search (MIPS) problem (Shrivastava & Li, 2014), for which sublinear-time hashing based methods are available.

Example 1. *Let $G = (V, E)$ be a graph, and let $\phi(x, a) \in \{0, 1\}^{|E|}$ represent a matching and $\theta \in \mathbb{R}^{|E|}$ be a vector of edge weights. The problem of finding the maximum-weight matching for a given set of edge weights can be written as a linear optimization problem of the form in Eq. (2), and Edmonds' algorithm (Edmonds, 1965) can be used to find the maximum-weight matching in $O(|V|^2 \cdot |E|)$ time.*

Other combinatorial problems that admit polynomial-time action optimization oracles include the maximum-weight spanning tree problem, the assignment problem, and others (Awerbuch & Kleinberg, 2008; Cesa-Bianchi & Lugosi, 2012).

Action representation. We define $b_{\mathcal{A}}$ as the number of bits used to represent actions in $\mathcal{A}$, which is always upper bounded by $O(\log|\mathcal{A}|)$ for finite action sets, and by $O(d)$ for actions that can be represented as vectors in $\mathbb{R}^d$. Tighter bounds are possible with additional structual assumptions. Since representing actions is a minimal assumption, we hide the dependence on $b_{\mathcal{A}}$ in big-O notation for our runtime and memory analysis.

2.2. Additional Notation

We adopt non-asymptotic big-oh notation: For functions $f, g : \mathcal{Z} \to \mathbb{R}_+$, we write $f = O(g)$ (resp. $f = \Omega(g)$) if there exists a constant $C > 0$ such that $f(z) \leq Cg(z)$ (resp. $f(z) \geq Cg(z)$) for all $z \in \mathcal{Z}$. We write $f = \widetilde{O}(g)$ if $f = O(g \cdot \mathrm{polylog}(T))$, $f = \widetilde{\Omega}(g)$ if $f = \Omega(g/\mathrm{polylog}(T))$. We use $\lesssim$ only in informal statements to highlight salient elements of an inequality.

For a vector $z \in \mathbb{R}^d$, we let $\|z\|$ denote the euclidean norm. We define $\|z\|_W^2 := \langle z, Wz \rangle$ for a positive definite matrix $W \in \mathbb{R}^{d \times d}$. For an integer $n \in \mathbb{N}$, we let $[n]$ denote the set $\{1, \dots, n\}$. For a set $\mathcal{Z}$, we let $\Delta(\mathcal{Z})$ denote the set of all Radon probability measures over $\mathcal{Z}$. We let $\mathrm{conv}(\mathcal{Z})$ denote the set of all finitely supported convex combinations of elements in $\mathcal{Z}$. When $\mathcal{Z}$ is finite, we let $\mathrm{unif}(\mathcal{Z})$ denote the uniform distribution over all the elements in $\mathcal{Z}$. We let $\mathbb{I}_z \in \Delta(\mathcal{Z})$ denote the delta distribution on z. We use the convention $a \wedge b = \min\{a, b\}$ and $a \vee b = \max\{a, b\}$.

3. Warm-Up: Oracle-Efficient Algorithms with Uniform Exploration

In this section, we present our first result: an efficient algorithm based on uniform exploration over a representative basis (SpannerGreedy; Algorithm 1). This algorithm achieves computational efficiency by taking advantage of an online regression oracle, but its regret bound has sub-optimal dependence on T. Beyond being practically useful in its own right, this result serves as a warm-up for Section 4.

Our algorithm is based on exploration with a *G-optimal design* for the embedding ϕ, which is a distribution over actions that minimizes a certain notion of worse-case variance (Kiefer & Wolfowitz, 1960; Atwood, 1969).

Definition 2 (G-optimal design)**.** *Let a set $\mathcal{Z} \subseteq \mathbb{R}^d$ be given. A distribution $q \in \Delta(\mathcal{Z})$ is said to be a G-optimal design with approximation factor $C_{\mathrm{opt}} \geq 1$ if*

$$\sup_{z \in \mathcal{Z}} \|z\|^2_{V(q)^{-1}} \leq C_{\mathrm{opt}} \cdot d,$$

where $V(q) := \mathbb{E}_{z \sim q}\left[zz^\top\right]$.

The following classical result guarantees existence of a G-optimal design.

Lemma 1 (Kiefer & Wolfowitz (1960))**.** *For any compact set $\mathcal{Z} \subseteq \mathbb{R}^d$, there exists an optimal design with $C_{\mathrm{opt}} = 1$.*

Algorithm 1 SpannerGreedy

Input: Exploration parameter $\varepsilon \in (0, 1]$, online regression oracle $\mathbf{Alg}_{\mathsf{Sq}}$, action optimization oracle $\mathbf{Alg}_{\mathsf{Opt}}$.

for $t = 1, 2, \dots, T$ **do**
- Observe context x_t.
- Receive $\widehat{f}_t = f_{\widehat{g}_t}$ from regression oracle $\mathbf{Alg}_{\mathsf{Sq}}$.
- Get $\widehat{a}_t \leftarrow \arg\max_{a \in \mathcal{A}} \langle \phi(x_t, a), \widehat{g}_t(x_t) \rangle$.
- Call subroutine to compute C_{opt}-approximate optimal design $q_t^{\mathrm{opt}} \in \Delta(\mathcal{A})$ for set $\{\phi(x_t, a)\}_{a \in \mathcal{A}}$.
  `// See Algorithm 5 for efficient solver.`
- Define $p_t := \varepsilon \cdot q_t^{\mathrm{opt}} + (1 - \varepsilon) \cdot \mathbb{I}_{\widehat{a}_t}$.
- Sample $a_t \sim p_t$ and observe reward $r_t(a_t)$.
- Update oracle $\mathbf{Alg}_{\mathsf{Sq}}$ with $(x_t, a_t, r_t(a_t))$.

Algorithm 1 uses optimal design as a basis for exploration: At each round, the learner obtains an estimator $\widehat{f}_t$ from the regression oracle $\mathbf{Alg}_{\mathsf{Sq}}$, then appeals to a subroutine to compute an (approximate) G-optimal design $q_t^{\mathrm{opt}} \in \Delta(\mathcal{A})$ for the action embedding $\{\phi(x_t, a)\}_{a\in\mathcal{A}}$. Fix an exploration parameter $\varepsilon > 0$, the algorithm then samples an action $a \sim q_t^{\mathrm{opt}}$ from the optimal design with probability ε ("exploration"), or plays the greedy action $\widehat{a}_t := \arg\max_{a\in\mathcal{A}} \widehat{f}_t(x_t, a)$ with probability $1-\varepsilon$ ("exploitation"). Algorithm 1 is efficient whenever an approximate optimal design can be computed efficiently, which can be achieved using Algorithm 5. We defer a detailed discussion of efficiency for a moment, and first state the main regret bound for the algorithm.

Theorem 1. *With a C_{opt}-approximate optimal design subroutine and an appropriate choice for $\varepsilon \in (0, 1]$, Algorithm 1, with probability at least $1-\delta$, enjoys regret*

$$\mathbf{Reg}_{\mathsf{CB}}(T) = O\Big((C_{\mathrm{opt}} \cdot d)^{1/3} T^{2/3} (\mathbf{Reg}_{\mathsf{Sq}}(T) + \log(\delta^{-1}))^{1/3}\Big).$$

In particular, when invoked with Algorithm 5 (with $C = 2$) as a subroutine, the algorithm enjoys regret

$$\mathbf{Reg}_{\mathsf{CB}}(T) = O\Big(d^{2/3} T^{2/3} (\mathbf{Reg}_{\mathsf{Sq}}(T) + \log(\delta^{-1}))^{1/3}\Big).$$

and has per-round runtime $O(\mathcal{T}_{\mathsf{Sq}} + \mathcal{T}_{\mathsf{Opt}} \cdot d^2 \log d + d^4 \log d)$ and maximum memory $O(\mathcal{M}_{\mathsf{Sq}} + \mathcal{M}_{\mathsf{Opt}} + d^2)$.

Computational efficiency. The computational efficiency of Algorithm 1 hinges on the ability to efficiently compute an approximate optimal design (or, by convex duality, the John ellipsoid (John, 1948)) for the set $\{\phi(x_t, a)\}_{a\in\mathcal{A}}$. All off-the-shelf optimal design solvers that we are aware of require solving quadratic maximization subproblems, which in general cannot be reduced to a linear optimization oracle (Definition 1). While there are some special cases where efficient solvers exist (e.g., when $\mathcal{A}$ is a polytope (Cohen et al. (2019) and references therein)), computing an exact optimal design is NP-hard in general (Grötschel et al., 2012; Summa et al., 2014). To overcome this issue, we use the notion of a *barycentric spanner*, which acts as an approximate optimal design and can be computed efficiently using an action optimization oracle.

Definition 3 (Awerbuch & Kleinberg (2008))**.** *Let a compact set $\mathcal{Z} \subseteq \mathbb{R}^d$ of full dimension be given. For $C \geq 1$, a subset of points $\mathcal{S} = \{z_1, \ldots, z_d\} \subseteq \mathcal{Z}$ is said to be a C-approximate barycentric spanner for $\mathcal{Z}$ if every point $z \in \mathcal{Z}$ can be expressed as a weighted combination of points in $\mathcal{S}$ with coefficients in $[-C, C]$.*

The following result shows that any barycentric spanner yields an approximate optimal design.

Lemma 2. *If $\mathcal{S} = \{z_1, \ldots, z_d\}$ is a C-approximate barycentric spanner for $\mathcal{Z} \subseteq \mathbb{R}^d$, then $q := \mathrm{unif}(\mathcal{S})$ is a $(C^2 \cdot d)$-approximate optimal design.*

Using an algorithm introduced by Awerbuch & Kleinberg (2008), one can efficiently compute the C-approximate barycentric spanner for the set $\{\phi(x, a)\}_{a\in\mathcal{A}}$ using$O(d^2 \log_C d)$ an action optimization oracle; their method is restated stated as Algorithm 5 in Appendix B.

Key features of Algorithm 1. While the regret bound for Algorithm 1 scales with $T^{2/3}$, which is not optimal, this result constitutes the first computationally efficient algorithm for contextual bandits with linearly structured actions and general function approximation. Additional features include:

- *Simplicity and practicality.* Appealing to uniform exploration makes Algorithm 1 easy to implement and highly practical. In particular, in the case where the action embedding does not depend on the context (i.e., $\phi(x, a) = \phi(a)$) an approximate design can be precomputed and reused, reducing the per-round runtime to $\widetilde{O}(\mathcal{T}_{\mathsf{Sq}} + \mathcal{T}_{\mathsf{Opt}})$ and the maximum memory to $O(\mathcal{M}_{\mathsf{Sq}} + d)$.
- *Lifting optimal design to contextual bandits.* Previous bandit algorithms based on optimal design are limited to the non-contextual setting, and to pure exploration. Our result highlights for the first time that optimal design can be efficiently combined with general function approximation.

Proof sketch for Theorem 1. To analyze Algorithm 1, we follow a recipe introduced by Foster & Rakhlin (2020); Foster et al. (2021b) based on the *Decision-Estimation Coefficient* (DEC),[1] defined as $\mathsf{dec}_\gamma(\mathcal{F}) := \sup_{\widehat{f}\in\mathrm{conv}(\mathcal{F}), x\in\mathcal{X}} \mathsf{dec}_\gamma(\mathcal{F}; \widehat{f}, x)$, where

$$\mathsf{dec}_\gamma(\mathcal{F}; \widehat{f}, x) := \inf_{p\in\Delta(\mathcal{A})} \sup_{a^\star\in\mathcal{A}} \sup_{f^\star\in\mathcal{F}} \mathbb{E}_{a\sim p}\Big[f^\star(x, a^\star) - f^\star(x, a) - \gamma \cdot (\widehat{f}(x, a) - f^\star(x, a))^2\Big]. \tag{3}$$

Foster et al. (2021b) consider a meta-algorithm which, at each round t, (i) computes $\widehat{f}_t$ by appealing to a regression oracle, (ii) computes a distribution $p_t \in \Delta(\mathcal{A})$ that solves the minimax problem in Eq. (3) with x_t and $\widehat{f}_t$ plugged in, and (iii) chooses the action a_t by sampling from this

[1] The original definition of the Decision-Estimation Coefficient in Foster et al. (2021b) uses Hellinger distance rather than squared error. The squared error version we consider here leads to tighter guarantees for bandit problems where the mean rewards serve as a sufficient statistic.

distribution. One can show (Lemma 7 in Appendix B) that for any $\gamma > 0$, this strategy enjoys the following regret bound:

$$\mathbf{Reg}_{\mathsf{CB}}(T) \lesssim T \cdot \mathsf{dec}_{\gamma}(\mathcal{F}) + \gamma \cdot \mathbf{Reg}_{\mathsf{Sq}}(T), \qquad (4)$$

More generally, if one computes a distribution that does not solve Eq. (3) exactly, but instead certifies an upper bound on the DEC of the form $\mathsf{dec}_{\gamma}(\mathcal{F}) \leq \overline{\mathsf{dec}}_{\gamma}(\mathcal{F})$, the same result holds with $\mathsf{dec}_{\gamma}(\mathcal{F})$ replaced by $\overline{\mathsf{dec}}_{\gamma}(\mathcal{F})$. Algorithm 1 is a special case of this meta-algorithm, so to bound the regret it suffices to show that the exploration strategy in the algorithm certifies a bound on the DEC.

Lemma 3. *For any $\gamma \geq 1$, by choosing $\varepsilon = \sqrt{C_{\mathrm{opt}} \cdot d/4\gamma} \wedge 1$, the exploration strategy in Algorithm 1 certifies that $\mathsf{dec}_{\gamma}(\mathcal{F}) = O(\sqrt{C_{\mathrm{opt}} \cdot d/\gamma})$.*

Using Lemma 3, one can upper bound the first term in Eq. (4) by $O(T\sqrt{C_{\mathrm{opt}} d/\gamma})$. The regret bound in Theorem 1 follows by choosing γ to balance the two terms.

4. Efficient, Near-Optimal Algorithms

In this section we present SpannerIGW (Algorithm 2), an efficient algorithm with $\widetilde{O}(\sqrt{T})$ regret (Algorithm 2). We provide the algorithm and statistical guarantees in Section 4.1, then discuss computational efficiency in Section 4.2.

4.1. Algorithm and Statistical Guarantees

Building on the approach in Section 3, SpannerIGW uses the idea of exploration with an optimal design. However, in order to achieve $\sqrt{T}$ regret, we combine optimal design with the *inverse gap weighting* (IGW) technique. previously used in the finite-action contextual bandit setting (Abe & Long, 1999; Foster & Rakhlin, 2020).

Recall that for finite-action contextual bandits, the inverse gap weighting technique works as follows. Given a context x_t and estimator $\widehat{f}_t$ from the regression oracle $\mathbf{Alg}_{\mathsf{Sq}}$, we assign a distribution to actions in $\mathcal{A}$ via the rule

$$p_t(a) := \frac{1}{\lambda + \gamma \cdot \left(\widehat{f}_t(x_t, \widehat{a}_t) - \widehat{f}_t(x_t, a)\right)},$$

where $\widehat{a}_t := \arg\max_{a \in \mathcal{A}} \widehat{f}_t(x_t, a)$ and $\lambda > 0$ is chosen such that $\sum_a p_t(a) = 1$. This strategy certifies that $\mathsf{dec}_{\gamma}(\mathcal{F}; \widehat{f}_t, x_t) \leq \frac{|\mathcal{A}|}{\gamma}$, which leads to regret $O\big(\sqrt{|\mathcal{A}| T \cdot \mathbf{Reg}_{\mathsf{Sq}}(T)}\big)$. While this is essentially optimal for the finite-action setting, the linear dependence on $|\mathcal{A}|$ makes it unsuitable for the large-action setting we consider.

To lift the IGW strategy to the large-action setting, Algorithm 2 combines it with optimal design with respect to a *reweighted embedding*. Let $\widehat{f} \in \mathcal{F}$ be given. For each action $a \in \mathcal{A}$, we define a reweighted embedding via

$$\bar{\phi}(x, a) := \frac{\phi(x, a)}{\sqrt{1 + \eta\Big(\widehat{f}(x, \widehat{a}) - \widehat{f}(x, a)\Big)}}, \qquad (5)$$

where $\widehat{a} := \arg\max_{a \in \mathcal{A}} \widehat{f}(x, a)$ and $\eta > 0$ is a reweighting parameter to be tuned later. This reweighting is *action-dependent* since $\widehat{f}(x, a)$ term appears on the denominator. Within Algorithm 2, we compute a new reweighted embedding at each round $t \in [T]$ using $\widehat{f}_t = f_{\widehat{g}_t}$, the output of the regression oracle $\mathbf{Alg}_{\mathsf{Sq}}$.

Algorithm 2 proceeds by computing an optimal design $q_t^{\mathrm{opt}} \in \Delta(\mathcal{A})$ with respect to the reweighted embedding defined in Eq. (5). The algorithm then creates a distribution $q_t := \frac{1}{2} q_t^{\mathrm{opt}} + \frac{1}{2} \mathbb{I}_{\widehat{a}_t}$ by mixing the optimal design with a delta mass at the greedy action $\widehat{a}_t$. Finally, in Eq. (6), the algorithm computes an augmented version of the inverse gap weighting distribution by reweighting according to q_t. This approach certifies the following bound on the Decision-Estimation Coefficient.

Lemma 4. *For any $\gamma > 0$, by setting $\eta = \gamma/(C_{\mathrm{opt}} \cdot d)$, the exploration strategy used in Algorithm 2 certifies that $\mathsf{dec}_{\gamma}(\mathcal{F}) = O(C_{\mathrm{opt}} \cdot d/\gamma)$.*

This lemma shows that the reweighted IGW strategy enjoys the best of both worlds: By leveraging optimal design, we ensure good coverage for all actions, leading to $O(d)$ (rather than $O(|\mathcal{A}|)$) scaling, and by leveraging inverse gap weighting, we avoid excessive exploration, leading $O(1/\gamma)$ rather than $O(1/\sqrt{\gamma})$ scaling. Combining this result with Lemma 7 leads to our main regret bound for SpannerIGW.

Theorem 2. *Let $\delta \in (0, 1)$ be given. With a C_{opt}-approximate optimal design subroutine and an appropriate choice for $\gamma > 0$, Algorithm 2 ensures that with probability at least $1 - \delta$,*

$$\mathbf{Reg}_{\mathsf{CB}}(T) = O\left(\sqrt{C_{\mathrm{opt}} \cdot d\, T \left(\mathbf{Reg}_{\mathsf{Sq}}(T) + \log(\delta^{-1})\right)}\right).$$

In particular, when invoked with Algorithm 3 (with $C = 2$) as a subroutine, the algorithm has

$$\mathbf{Reg}_{\mathsf{CB}}(T) = O\left(d\sqrt{T\left(\mathbf{Reg}_{\mathsf{Sq}}(T) + \log(\delta^{-1})\right)}\right),$$

and has per-round runtime $O(\mathcal{T}_{\mathsf{Sq}} + (\mathcal{T}_{\mathsf{Opt}} \cdot d^3 + d^4) \cdot \log^2\left(\frac{T}{r}\right))$ and the maximum memory $O(\mathcal{M}_{\mathsf{Sq}} + \mathcal{M}_{\mathsf{Opt}} + d^2 + d\log\left(\frac{T}{r}\right))$.

Algorithm 2 is the first computationally efficient algorithm with $\sqrt{T}$-regret for contextual bandits with general function approximation and linearly structured action spaces. In what follows, we show how to leverage the action optimization oracle (Definition 1) to achieve this efficiency.

Algorithm 2 SpannerIGW

Input: Exploration parameter $\gamma > 0$, online regression oracle $\mathbf{Alg}_{\mathsf{Sq}}$, action optimization oracle $\mathbf{Alg}_{\mathsf{Opt}}$.

Define $\eta := \frac{\gamma}{C_{\mathrm{opt}} \cdot d}$.
for $t = 1, 2, \ldots, T$ **do**
  Observe context x_t.
  Receive $\widehat{f}_t = f_{\widehat{g}_t}$ from regression oracle $\mathbf{Alg}_{\mathsf{Sq}}$.
  Get $\widehat{a}_t \leftarrow \arg\max_{a \in \mathcal{A}} \langle \phi(x_t, a), \widehat{g}_t(x_t) \rangle$.
  Call subroutine to compute C_{opt}-approximate optimal design $q_t^{\mathrm{opt}} \in \Delta(\mathcal{A})$ for reweighted embedding $\{\bar{\phi}(x_t, a)\}_{a \in \mathcal{A}}$ (Eq. (5) with $\widehat{f} = \widehat{f}_t$).
  `// See Algorithm 3 for efficient solver.`
  Define $q_t := \frac{1}{2} q_t^{\mathrm{opt}} + \frac{1}{2} \mathbb{I}_{\widehat{a}_t}$.
  For each $a \in \mathrm{supp}(q_t)$, define

$$p_t(a) := \frac{q_t(a)}{\lambda + \eta\Big(\widehat{f}_t(x_t, \widehat{a}_t) - \widehat{f}_t(x_t, a)\Big)}, \qquad (6)$$

  where $\lambda \in [\frac{1}{2}, 1]$ is chosen so that $\sum_{a \in \mathrm{supp}(q_t)} p_t(a) = 1$.
  Sample $a_t \sim p_t$ and observe reward $r_t(a_t)$.
  Update $\mathbf{Alg}_{\mathsf{Sq}}$ with $(x_t, a_t, r_t(a_t))$.

4.2. Computational Efficiency

The computational efficiency of Algorithm 2 hinges on the ability to efficiently compute an optimal design. As with Algorithm 1, we address this issue by appealing to the notion of a barycentric spanner, which serves as an approximate optimal design. However, compared to Algorithm 1, a substantial additional challenge is that Algorithm 2 requires an approximate optimal design for the *reweighted* embeddings. Since the reweighting is action-dependent, the action optimization oracle $\mathbf{Alg}_{\mathsf{Opt}}$ cannot be directly applied to optimize over the reweighted embeddings, which prevents us from appealing to an out-of-the-box solver (Algorithm 5) in the same fashion as the prequel

To address the challenges above, we introduce ReweightedSpanner (Algorithm 3), a barycentric spanner computation algorithm which is tailored to the reweighted embedding $\bar{\phi}$. To describe the algorithm, let us introduce some additional notation. For a set $\mathcal{S} \subseteq \mathcal{A}$ of d actions, we let $\det(\bar{\phi}(x, \mathcal{S}))$ denote the determinant of the d-by-d matrix whose columns are $\{\bar{\phi}(x, a)\}_{a \in \mathcal{A}}$. ReweightedSpanner adapts the barycentric spanner computation approach of Awerbuch & Kleinberg (2008), which aims to identify a subset $\mathcal{S} \subseteq \mathcal{A}$ with $|\mathcal{S}| = d$ that approximately maximizes $|\det(\bar{\phi}(x, \mathcal{S}))|$. The key feature of ReweightedSpanner is a subroutine, IGW-ArgMax (Algorithm 4), which implements an (approximate) action optimization oracle for

Algorithm 3 ReweightedSpanner

Input: Context $x \in \mathcal{X}$, oracle prediction $\widehat{g}(x) \in \mathbb{R}^d$, action $\widehat{a} := \arg\max_{a \in \mathcal{A}} \langle \phi(x, a), \widehat{g}(x) \rangle$, reweighting parameter $\eta > 0$, approximation factor $C > \sqrt{2}$, initial set $\mathcal{S} = (a_1, \ldots, a_d)$ with $|\det(\phi(x, \mathcal{S}))| \geq r^d$ for $r \in (0, 1)$.

while not break **do**
  for $i = 1, \ldots, d$ **do**
    Compute $\theta \in \mathbb{R}^d$ representing linear function $\bar{\phi}(x, a) \mapsto \det(\bar{\phi}(x, \mathcal{S}_i(a)))$, where $\mathcal{S}_i(a) := (a_1, \ldots, a_{i-1}, a, a_{i+1}, \ldots, a_d)$. `// $\bar{\phi}$ is computed from $f_{\widehat{g}}$, $\widehat{a}$, and $\eta$ via Eq. (5).`
    Get $a \leftarrow$ IGW-ArgMax$(\theta; x, \widehat{g}(x), \eta, r)$. `// Algorithm 4.`
    if $\big|\det(\bar{\phi}(x, \mathcal{S}_i(a)))\big| \geq \frac{\sqrt{2}C}{2} \big|\det(\bar{\phi}(x, \mathcal{S}))\big|$ **then**
      Update $a_i \leftarrow a$.
      continue to line 2.
  break
return C-approximate barycentric spanner $\mathcal{S}$.

the reweighted embedding:

$$\arg\max_{a \in \mathcal{A}} \big\langle \bar{\phi}(x, a), \theta \big\rangle \qquad (7)$$

IGW-ArgMax uses line search reduce the problem in Eq. (7) to a sequence of linear optimization problems with respect to the *unweighted* embeddings, each of which can be solved using $\mathbf{Alg}_{\mathsf{Opt}}$. This yields the following guarantee for Algorithm 3.

Theorem 3. *Suppose that Algorithm 3 is invoked with parameters $\eta > 0$, $r \in (0, 1)$, and $C > \sqrt{2}$, and that the initialization set $\mathcal{S}$ satisfies $|\det(\phi(x, \mathcal{S}))| \geq r^d$. Then the algorithm returns a C-approximate barycentric spanner with respect to the reweighted embedding set $\{\bar{\phi}(x, a)\}_{a \in \mathcal{A}}$, and does so with $O((\mathcal{T}_{\mathsf{Opt}} \cdot d^3 + d^4) \cdot \log^2(e \vee \frac{\eta}{r}))$ runtime and $O(\mathcal{M}_{\mathsf{Opt}} + d^2 + d \log(e \vee \frac{\eta}{r}))$ memory.*

We refer to Appendix D.1 for self-contained analysis of IGW-ArgMax.

On the initialization requirement. The runtime for Algorithm 3 scales with $\log(r^{-1})$, where $r \in (0, 1)$ is such that $\det(\phi(x, \mathcal{S})) \geq r^d$ for the initial set $\mathcal{S}$. In Appendix D.3, we provide computationally efficient algorithms for initialization under various assumptions on the action space.

5. Empirical Results

In this section we investigate the empirical performance of SpannerGreedy and SpannerIGW through three experiments. First, we compare the spanner-based algorithms to state-of-the art finite-action algorithms on a large-action dataset; this

Algorithm 4 IGW-ArgMax

Input: Linear parameter $\theta \in \mathbb{R}^d$, context $x \in \mathcal{X}$, oracle prediction $\widehat{g}(x) \in \mathbb{R}^d$, reweighting parameter $\eta > 0$, initialization constant $r \in (0, 1)$.

Define $N := \lceil d \log_{\frac{4}{3}}(\frac{2\eta+1}{r}) \rceil$.
Define $\mathcal{E} := \{(\frac{3}{4})^i\}_{i=1}^N \cup \{-(\frac{3}{4})^i\}_{i=1}^N$.
Initialize $\widehat{\mathcal{A}} = \emptyset$.
for each $\varepsilon \in \mathcal{E}$ **do**
  Compute $\bar{\theta} \leftarrow 2\varepsilon\theta + \varepsilon^2\eta \cdot \widehat{g}(x)$.
  Get $a \leftarrow \arg\max_{a \in \mathcal{A}} \langle \phi(x, a), \bar{\theta} \rangle$; add a to $\widehat{\mathcal{A}}$.
return $\arg\max_{a \in \widehat{\mathcal{A}}} \langle \bar{\phi}(x, a), \theta \rangle^2$ // $\widetilde{O}(d)$ candidates.

experiment features nonlinear, learned context embeddings $g \in \mathcal{G}$. Next, we study the impact of redundant actions on the statistical performance of said algorithms. Finally, we experiment with a large-scale large-action contextual bandit benchmark, where we find that the spanner-based methods exhibit excellent performance.

Preliminaries. We conduct experiments on three datasets, whose details are summarized in Table 1. oneshotwiki (Singh et al., 2012; Vasnetsov, 2018) is a named-entity recognition task where contexts are text phrases preceding and following the mention text, and where actions are text phrases corresponding to the concept names. amazon-3m (Bhatia et al., 2016) is an extreme multi-label dataset whose contexts are text phrases corresponding to the title and description of an item, and whose actions are integers corresponding to item tags. Actions are embedded into $\mathbb{R}^d$ with d specified in Table 1. We construct binary rewards for each dataset, and report 90% bootstrap CIs of the rewards in the experiments. We defer other experimental details to Appendix E.1. Code to reproduce all results is available at https://github.com/pmineiro/linrepcb.

Table 1: Datasets used for experiments.

Dataset	T	$\lvert\mathcal{A}\rvert$	d
oneshotwiki-311	622000	311	50
oneshotwiki-14031	2806200	14031	50
amazon-3m	1717899	2812281	800

Comparison with finite-action baselines. We compare SpannerGreedy and SpannerIGW with their finite-action counterparts ε-Greedy and SquareCB (Foster & Rakhlin, 2020) on the oneshotwiki-14031 dataset. We consider *bilinear models* in which regression functions take the form $f(x, a) = \langle \phi(a), Wx \rangle$ where W is a matrix of learned parameters; the *deep models* of the form $f(x, a) = \langle \phi(a), W\bar{g}(x) \rangle$, where $\bar{g}$ is a learned two-layer neural network and W contains learned parameters as before.[2] Table 2 presents our results. We find that SpannerIGW performs best, and that both spanner-based algorithms either tie or exceed their finite-action counterparts. In addition, we find that working with deep models uniformly improves performance for all methods. We refer to Table 4 in Appendix E.3 for timing information.

Table 2: Comparison on oneshotwiki-14031. Values are the average progressive-validation reward, scaled by 1000. We include the performance of the best constant predictor and supervised learning as a baseline and skyline respectively.

Algorithm	Regression Function	
	Bilinear	Deep
best constant	0.07127	
ε-Greedy	$[5.00, 6.27]$	$[7.15, 8.52]$
SpannerGreedy	$[6.29, 7.08]$	$[6.67, 8.30]$
SquareCB	$[7.57, 8.59]$	$[10.4, 11.3]$
SpannerIGW	$[8.84, 9.68]$	$[11.2, 12.2]$
supervised	$[31.2, 31.3]$	$[36.7, 36.8]$

Impact of redundancy. Finite-action contextual bandit algorithms can explore excessively in the presence of redundant actions. To evaluate performance in the face of redundancy, we augment oneshotwiki-311 by duplicating action the final action. Table 3 displays the performance of SpannerIGW and its finite-action counterpart, SquareCB, with a varying number of duplicates. We find that SpannerIGW is completely invariant to duplicates (in fact, the algorithm produces numerically identical output when the random seed is fixed), but SquareCB is negatively impacted and over-explores the duplicated action. SpannerGreedy and ε-Greedy behave analogously (not shown).

Table 3: Redundancy study on oneshotwiki-311. Values are the average progressive reward per example scaled by 100.

Duplicates	SpannerIGW	SquareCB
0	$[12.6, 13.0]$	$[12.2, 12.6]$
16	$[12.6, 13.0]$	$[12.1, 12.4]$
256	$[12.6, 13.0]$	$[10.2, 10.6]$
1024	$[12.6, 13.0]$	$[8.3, 8.6]$

Large scale exhibition. We conduct a large scale experiment using the amazon-3m dataset. Following Sen et al. (2021), we study the top-k setting where k actions are selected at each round. Out of the total number of actions sampled, we let r denote the number of actions sampled for exploration. We apply SpannerGreedy for this dataset and

[2] Also see Appendix E.1 for details.

consider regression functions similar to the deep models discussed before. The setting $(k = 1)$ corresponds to running our algorithm unmodified, and $(k = 5, r = 3)$ corresponds to selecting 5 actions per round and using 3 exploration slots. Figure 1 in Appendix E.4 displays the results. For $(k = 1)$ the final CI is $[0.1041, 0.1046]$, and for $(k = 5, r = 3)$ the final CI is $[0.438, 0.440]$.

In the setup with $(k = 5, r = 3)$, our results are directly comparable to Sen et al. (2021), who evaluated a tree-based contextual bandit method on the same dataset. The best result from Sen et al. (2021) achieves roughly 0.19 reward with $(k = 5, r = 3)$, which we exceed by a factor of 2. This indicates that our use of embeddings provides favorable inductive bias for this problem, and underscores the broad utility of our techniques (which leverage embeddings). For $(k = 5, r = 3)$, our inference time on a commodity CPU with batch size 1 is 160ms per example, which is slower than the time of 7.85ms per example reported in Sen et al. (2021).

6. Discussion

We provide the first efficient algorithms for contextual bandits with continuous, linearly structured action spaces and general-purpose function approximation. We highlight some natural directions for future research below.

- **Efficient algorithms for nonlinear action spaces.** Our algorithms take advantage of linearly structured action spaces by appealing to optimal design. Can we develop computationally efficient methods for contextual bandits with nonlinear dependence on the action space?

- **Reinforcement learning.** The contextual bandit problem is a special case of the reinforcement learning problem with horizon one. Given our positive results in the contextual bandit setting, a natural next step is to extend our methods to reinforcement learning problems with large action/decision spaces. For example, Foster et al. (2021b) build on our computational tools to provide efficient algorithms for reinforcement learning with bilinear classes.

Beyond these directions, natural domains in which to extend our techniques include pure exploration and off-policy learning with linearly structured actions.

References

Abbasi-Yadkori, Y., Pál, D., and Szepesvári, C. Improved algorithms for linear stochastic bandits. In *NIPS*, volume 11, pp. 2312–2320, 2011.

Abe, N. and Long, P. M. Associative reinforcement learning using linear probabilistic concepts. In *ICML*, pp. 3–11. Citeseer, 1999.

Agarwal, A., Dudík, M., Kale, S., Langford, J., and Schapire, R. Contextual bandit learning with predictable rewards. In *Artificial Intelligence and Statistics*, pp. 19–26. PMLR, 2012.

Agarwal, A., Hsu, D., Kale, S., Langford, J., Li, L., and Schapire, R. Taming the monster: A fast and simple algorithm for contextual bandits. In *International Conference on Machine Learning*, pp. 1638–1646. PMLR, 2014.

Agarwal, A., Bird, S., Cozowicz, M., Hoang, L., Langford, J., Lee, S., Li, J., Melamed, D., Oshri, G., Ribas, O., Sen, S., and Slivkins, A. Making contextual decisions with low technical debt. *arXiv:1606.03966*, 2016.

Agrawal, S. and Goyal, N. Thompson sampling for contextual bandits with linear payoffs. In *International Conference on Machine Learning*, pp. 127–135. PMLR, 2013.

Atwood, C. L. Optimal and efficient designs of experiments. *The Annals of Mathematical Statistics*, pp. 1570–1602, 1969.

Awerbuch, B. and Kleinberg, R. Online linear optimization and adaptive routing. *Journal of Computer and System Sciences*, 74(1):97–114, 2008.

Bergstra, J. and Bengio, Y. Random search for hyper-parameter optimization. *Journal of machine learning research*, 13(2), 2012.

Bhatia, K., Dahiya, K., Jain, H., Kar, P., Mittal, A., Prabhu, Y., and Varma, M. The extreme classification repository: Multi-label datasets and code, 2016. URL http://manikvarma.org/downloads/XC/XMLRepository.html.

Bhatnagar, S., Sutton, R. S., Ghavamzadeh, M., and Lee, M. Natural Actor–Critic algorithms. *Automatica*, 45(11): 2471–2482, 2009.

Bietti, A., Agarwal, A., and Langford, J. A contextual bandit bake-off. *Journal of Machine Learning Research*, 22(133):1–49, 2021.

Bubeck, S., Cesa-Bianchi, N., and Kakade, S. M. Towards minimax policies for online linear optimization with bandit feedback. In *Conference on Learning Theory*, pp. 41–1. JMLR Workshop and Conference Proceedings, 2012.

Cai, W., Grossman, J., Lin, Z. J., Sheng, H., Wei, J. T.-Z., Williams, J. J., and Goel, S. Bandit algorithms to personalize educational chatbots. *Machine Learning*, pp. 1–30, 2021.

Cao, T. and Krishnamurthy, A. Disagreement-based combinatorial pure exploration: Sample complexity bounds and an efficient algorithm. In *Conference on Learning Theory*, pp. 558–588. PMLR, 2019.

Cesa-Bianchi, N. and Lugosi, G. Combinatorial bandits. *Journal of Computer and System Sciences*, 78(5):1404–1422, 2012.

Chen, L., Gupta, A., Li, J., Qiao, M., and Wang, R. Nearly optimal sampling algorithms for combinatorial pure exploration. In *Conference on Learning Theory*, pp. 482–534. PMLR, 2017.

Chernozhukov, V., Demirer, M., Lewis, G., and Syrgkanis, V. Semi-parametric efficient policy learning with continuous actions. *Advances in Neural Information Processing Systems*, 32:15065–15075, 2019.

Chu, W., Li, L., Reyzin, L., and Schapire, R. Contextual bandits with linear payoff functions. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, pp. 208–214. JMLR Workshop and Conference Proceedings, 2011.

Cohen, M. B., Cousins, B., Lee, Y. T., and Yang, X. A near-optimal algorithm for approximating the John Ellipsoid. In *Conference on Learning Theory*, pp. 849–873. PMLR, 2019.

Dani, V. and Hayes, T. P. Robbing the bandit: Less regret in online geometric optimization against an adaptive adversary. In *SODA*, volume 6, pp. 937–943, 2006.

Dani, V., Hayes, T. P., and Kakade, S. M. Stochastic linear optimization under bandit feedback. *Conference on Learning Theory (COLT)*, 2008.

Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. Bert: Pre-training of deep bidirectional transformers for language understanding. In *NAACL-HLT (1)*, 2019.

Dudik, M., Hsu, D., Kale, S., Karampatziakis, N., Langford, J., Reyzin, L., and Zhang, T. Efficient optimal learning for contextual bandits. In *Proceedings of the Twenty-Seventh Conference on Uncertainty in Artificial Intelligence*, pp. 169–178, 2011.

Edmonds, J. Paths, trees, and flowers. *Canadian Journal of mathematics*, 17:449–467, 1965.

Fiez, T., Jain, L., Jamieson, K. G., and Ratliff, L. Sequential experimental design for transductive linear bandits. *Advances in neural information processing systems*, 32, 2019.

Foster, D. and Rakhlin, A. Beyond UCB: Optimal and efficient contextual bandits with regression oracles. In *International Conference on Machine Learning*, pp. 3199–3210. PMLR, 2020.

Foster, D., Agarwal, A., Dudik, M., Luo, H., and Schapire, R. Practical contextual bandits with regression oracles. In *International Conference on Machine Learning*, pp. 1539–1548. PMLR, 2018.

Foster, D., Rakhlin, A., Simchi-Levi, D., and Xu, Y. Instance-dependent complexity of contextual bandits and reinforcement learning: A disagreement-based perspective. In *Conference on Learning Theory*, pp. 2059–2059. PMLR, 2021a.

Foster, D. J. and Krishnamurthy, A. Efficient first-order contextual bandits: Prediction, allocation, and triangular discrimination. *Advances in Neural Information Processing Systems*, 34, 2021.

Foster, D. J., Gentile, C., Mohri, M., and Zimmert, J. Adapting to misspecification in contextual bandits. *Advances in Neural Information Processing Systems*, 33, 2020.

Foster, D. J., Kakade, S. M., Qian, J., and Rakhlin, A. The statistical complexity of interactive decision making. *arXiv preprint arXiv:2112.13487*, 2021b.

Grötschel, M., Lovász, L., and Schrijver, A. *Geometric algorithms and combinatorial optimization*, volume 2. Springer Science & Business Media, 2012.

Hazan, E. and Karnin, Z. Volumetric spanners: An efficient exploration basis for learning. *The Journal of Machine Learning Research*, 17(1):4062–4095, 2016.

Hazan, E., Agarwal, A., and Kale, S. Logarithmic regret algorithms for online convex optimization. *Machine Learning*, 69(2-3):169–192, 2007.

Ito, S., Hatano, D., Sumita, H., Takemura, K., Fukunaga, T., Kakimura, N., and Kawarabayashi, K.-I. Oracle-efficient algorithms for online linear optimization with bandit feedback. *Advances in Neural Information Processing Systems*, 32:10590–10599, 2019.

John, F. Extremum problems with inequalities as subsidiary conditions. *R. Courant Anniversary Volume*, pp. 187–204, 1948.

Jun, K.-S., Bhargava, A., Nowak, R., and Willett, R. Scalable generalized linear bandits: Online computation and hashing. *Advances in Neural Information Processing Systems*, 30, 2017.

Kassraie, P. and Krause, A. Neural contextual bandits without regret. In *International Conference on Artificial Intelligence and Statistics*, pp. 240–278. PMLR, 2022.

Katz-Samuels, J., Jain, L., Jamieson, K. G., et al. An empirical process approach to the union bound: Practical algorithms for combinatorial and linear bandits. *Advances in Neural Information Processing Systems*, 33, 2020.

Kiefer, J. and Wolfowitz, J. The equivalence of two extremum problems. *Canadian Journal of Mathematics*, 12: 363–366, 1960.

Krishnamurthy, A., Langford, J., Slivkins, A., and Zhang, C. Contextual bandits with continuous actions: Smoothing, zooming, and adapting. *Journal of Machine Learning Research*, 21(137):1–45, 2020.

Langford, J. and Zhang, T. The epoch-greedy algorithm for contextual multi-armed bandits. *Advances in neural information processing systems*, 20(1):96–1, 2007.

Lattimore, T. and Szepesvári, C. *Bandit algorithms*. Cambridge University Press, 2020.

Lattimore, T., Szepesvari, C., and Weisz, G. Learning with good feature representations in bandits and in RL with a generative model. In *International Conference on Machine Learning*, pp. 5662–5670. PMLR, 2020.

Lebret, R. and Collobert, R. Word embeddings through Hellinger PCA. In *Proceedings of the 14th Conference of the European Chapter of the Association for Computational Linguistics*, pp. 482–490, 2014.

Li, L., Chu, W., Langford, J., and Schapire, R. E. A contextual-bandit approach to personalized news article recommendation. In *Proceedings of the 19th international conference on World wide web*, pp. 661–670, 2010.

Mahabadi, S., Indyk, P., Gharan, S. O., and Rezaei, A. Composable core-sets for determinant maximization: A simple near-optimal algorithm. In *International Conference on Machine Learning*, pp. 4254–4263. PMLR, 2019.

Majzoubi, M., Zhang, C., Chari, R., Krishnamurthy, A., Langford, J., and Slivkins, A. Efficient contextual bandits with continuous actions. *Advances in Neural Information Processing Systems*, 33, 2020.

McMahan, H. B. and Blum, A. Online geometric optimization in the bandit setting against an adaptive adversary. In *International Conference on Computational Learning Theory*, pp. 109–123. Springer, 2004.

Meyer, C. D. *Matrix analysis and applied linear algebra*, volume 71. Siam, 2000.

Pan, F., Cai, Q., Tang, P., Zhuang, F., and He, Q. Policy gradients for contextual recommendations. In *The World Wide Web Conference*, pp. 1421–1431, 2019.

Reimers, N. and Gurevych, I. Sentence-BERT: Sentence embeddings using Siamese BERT-Networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, 11 2019. URL http://arxiv.org/abs/1908.10084.

Ruan, Y., Yang, J., and Zhou, Y. Linear bandits with limited adaptivity and learning distributional optimal design. In *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing*, pp. 74–87, 2021.

Sahni, S. Computationally related problems. *SIAM Journal on computing*, 3(4):262–279, 1974.

Schrijver, A. *Theory of linear and integer programming*. John Wiley & Sons, 1998.

Sen, R., Rakhlin, A., Ying, L., Kidambi, R., Foster, D., Hill, D. N., and Dhillon, I. S. Top-k extreme contextual bandits with arm hierarchy. In *International Conference on Machine Learning*, pp. 9422–9433. PMLR, 2021.

Sherman, J. and Morrison, W. J. Adjustment of an inverse matrix corresponding to a change in one element of a given matrix. *The Annals of Mathematical Statistics*, 21 (1):124–127, 1950.

Shrivastava, A. and Li, P. Asymmetric LSH (ALSH) for sublinear time maximum inner product search (MIPS). *Advances in neural information processing systems*, 27, 2014.

Simchi-Levi, D. and Xu, Y. Bypassing the monster: A faster and simpler optimal algorithm for contextual bandits under realizability. *Mathematics of Operations Research*, 2021.

Singh, S., Subramanya, A., Pereira, F., and McCallum, A. Wikilinks: A large-scale cross-document coreference corpus labeled via links to Wikipedia. Technical Report UM-CS-2012-015, University of Massachusetts, Amherst, 2012.

Soare, M., Lazaric, A., and Munos, R. Best-arm identification in linear bandits. *Advances in Neural Information Processing Systems*, 27, 2014.

Summa, M. D., Eisenbrand, F., Faenza, Y., and Moldenhauer, C. On largest volume simplices and subdeterminants. In *Proceedings of the twenty-sixth annual ACM-SIAM symposium on Discrete algorithms*, pp. 315–323. SIAM, 2014.

Tewari, A. and Murphy, S. A. From Ads to interventions: Contextual bandits in mobile health. In *Mobile Health*, pp. 495–517. Springer, 2017.

Vasnetsov, A. Oneshot-wikilinks. https://www.kaggle.com/generall/oneshotwikilinks, 2018.

Vovk, V. A game of prediction with expert advice. *Journal of Computer and System Sciences*, 56(2):153–173, 1998.

Wagenmaker, A., Katz-Samuels, J., and Jamieson, K. Experimental design for regret minimization in linear bandits. In *International Conference on Artificial Intelligence and Statistics*, pp. 3088–3096, 2021.

Williams, R. J. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3):229–256, 1992.

Xu, P., Wen, Z., Zhao, H., and Gu, Q. Neural contextual bandits with deep representation and shallow exploration. *arXiv preprint arXiv:2012.01780*, 2020.

Xu, Y. and Zeevi, A. Upper counterfactual confidence bounds: a new optimism principle for contextual bandits. *arXiv preprint arXiv:2007.07876*, 2020.

Yang, S., Ren, T., Shakkottai, S., Price, E., Dhillon, I. S., and Sanghavi, S. Linear bandit algorithms with sublinear time complexity. *arXiv preprint arXiv:2103.02729*, 2021.

Zhang, T. Feel-good thompson sampling for contextual bandits and reinforcement learning. *arXiv preprint arXiv:2110.00871*, 2021.

Zhang, W., Zhou, D., Li, L., and Gu, Q. Neural Thompson sampling. In *International Conference on Learning Representation (ICLR)*, 2021.

Zhou, D., Li, L., and Gu, Q. Neural contextual bandits with ucb-based exploration. In *International Conference on Machine Learning*, pp. 11492–11502. PMLR, 2020.

A. Additional Related Work

In this section we highlight some relevant lines of research not already discussed.

Efficient general-purpose contextual bandit algorithms. There is a long line of research on computationally efficient methods for contextual bandits with general function approximation, typically based on reduction to either cost-sensitive classification oracles (Langford & Zhang, 2007; Dudik et al., 2011; Agarwal et al., 2014) or regression oracles (Foster et al., 2018; Foster & Rakhlin, 2020; Simchi-Levi & Xu, 2021). Most of these works deal with a finite action spaces and have regret scaling with the number of actions, which is necessary without further structural assumptions (Agarwal et al., 2012). An exception is the works of Foster et al. (2020) and Xu & Zeevi (2020), both of which consider the same setting as the present paper. Both of the algorithms in these works require solving subproblems based on maximizing quadratic forms (which is NP-hard in general (Sahni, 1974)), and cannot directly take advantage of the linear optimization oracle we consider. Also related is the work of Zhang (2021), which proposes a posterior sampling-style algorithm for the setting we consider. This algorithm is not fully comparable computationally, as it requires sampling from specific posterior distribution; it is unclear whether this can be achieved in a provably efficient fashion.

Linear contextual bandits. The linear contextual bandit problem is a special case of our setting in which $g^\star(x) = \theta \in \mathbb{R}^d$ is constant (that is, the reward function only depends on the context through the feature map ϕ). The most well-studied families of algorithms for this setting are UCB-style algorithms and posterior sampling. With a well-chosen prior and posterior distribution, posterior sampling can be implemented efficiently (Agrawal & Goyal, 2013), but it is unclear how to efficiently adapt this approach to accomodate general function approximation. Existing UCB-type algorithms require solving sub-problems based on maximizing quadratic forms, which is NP-hard in general (Sahni, 1974). One line of research aims to make UCB efficient by using hashing-based methods (MIPS) to approximate the maximum inner product (Yang et al., 2021; Jun et al., 2017). These methods have runtime sublinear (but still polynomial) in the number of actions.

Non-contextual linear bandits. For the problem of *non-contextual* linear bandits (with either stochastic or adversarial rewards), there is a long line of research on efficient algorithms that can take advantage of linear optimization oracles (Awerbuch & Kleinberg, 2008; McMahan & Blum, 2004; Dani & Hayes, 2006; Dani et al., 2008; Bubeck et al., 2012; Hazan & Karnin, 2016; Ito et al., 2019); see also work on the closely related problem of combinatorial pure exploration (Chen et al., 2017; Cao & Krishnamurthy, 2019; Katz-Samuels et al., 2020; Wagenmaker et al., 2021). In general, it is not clear how to lift these techniques to contextual bandits with linearly-structured actions and general function approximation. We also mention that optimal design has been applied in the context of linear bandits, but these algorithms are restricted to the non-contextual setting (Lattimore & Szepesvári, 2020; Lattimore et al., 2020), or to pure exploration (Soare et al., 2014; Fiez et al., 2019). The only exception we are aware of is Ruan et al. (2021), who extend these developments to linear contextual bandits (i.e., where $g^\star(x) = \theta$), but critically use that contexts are stochastic.

Other approaches. Another line of research provides efficient contextual bandit methods under specific modeling assumptions on the context space or action space that differ from the ones we consider here. Zhou et al. (2020); Xu et al. (2020); Zhang et al. (2021); Kassraie & Krause (2022) provide generalizations of the UCB algorithm and posterior sampling based on the Neural Tangent Kernel (NTK). These algorithms can be used to learn context embeddings (i.e., $g(x)$) with general function approximation, but only lead to theoretical guarantees under strong RKHS-based assumptions. For large action spaces, these algorithms typically require enumeration over actions. Majzoubi et al. (2020) consider a setting with nonparametric action spaces and design an efficient tree-based learner; their guarantees, however, scale exponentially in the dimensionality of action space. Sen et al. (2021) provide heuristically-motivated but empirically-effective tree-based algorithms for contextual bandits with large action spaces, with theoretical guarantees when the actions satisfy certain tree-structured properties. Lastly, another empirically-successful approach is the policy gradient method (e.g., Williams (1992); Bhatnagar et al. (2009); Pan et al. (2019)). On the theoretical side, policy gradient methods do not address the issue of systematic exploration, and—to our knowledge—do not lead to provable guarantees for the setting considered in our paper.

B. Proofs and Supporting Results from Section 3

This section is organized as follows. We provide supporting results in Appendix B.1, then give the proof of Theorem 1 in Appendix B.2.

B.1. Supporting Results

B.1.1. Barycentric Spanner and Optimal Design

Algorithm 5 restates an algorithm of Awerbuch & Kleinberg (2008), which efficiently computes a barycentric spanner (Definition 3) given access to a linear optimization oracle (Definition 1). Recall that, for a set $\mathcal{S} \subset \mathcal{A}$ of d actions, the notation $\det(\bar{\phi}(x, \mathcal{S}))$ (resp. $\det(\phi(x, \mathcal{S}))$) denotes the determinant of the d-by-d matrix whose columns are the $\bar{\phi}$ (resp. ϕ) embeddings of actions.

Algorithm 5 Approximate Barycentric Spanner (Awerbuch & Kleinberg, 2008)

Input: Context $x \in \mathcal{X}$ and approximation factor $C > 1$.
1: **for** $i = 1, \dots, d$ **do**
2: Compute $\theta \in \mathbb{R}^d$ representing linear function $\phi(x, a) \mapsto \det(\phi(x, a_1), \dots, \phi(x, a_{i-1}), \phi(x, a), e_{i+1}, \dots, e_d)$.
3: Get $a_i \leftarrow \arg\max_{a \in \mathcal{A}} |\langle \phi(x, a), \theta \rangle|$.
4: Construct $\mathcal{S} = (a_1, \dots, a_d)$. `// Initial set of actions` $\mathcal{S} \subseteq \mathcal{A}$ `such that` $|\mathcal{S}| = d$ `and` $|\det(\phi(x, \mathcal{S}))| > 0$.
5: **while** not break **do**
6: **for** $i = 1, \dots, d$ **do**
7: Compute $\theta \in \mathbb{R}^d$ representing linear function $\phi(x, a) \mapsto \det(\phi(x, \mathcal{S}_i(a)))$, where $\mathcal{S}_i(a) := (a_1, \dots, a_{i-1}, a, a_{i+1}, \dots, a_d)$.
8: Get $a \leftarrow \arg\max_{a \in \mathcal{A}} |\langle \phi(x, a), \theta \rangle|$.
9: **if** $|\det(\phi(x, \mathcal{S}_i(a)))| \geq C |\det(\phi(x, \mathcal{S}))|$ **then**
10: Update $a_i \leftarrow a$.
11: **continue** to line 5.
12: **break**
13: **return** C-approximate barycentric spanner $\mathcal{S}$.

Lemma 5 (Awerbuch & Kleinberg (2008))**.** *For any $x \in \mathcal{X}$, Algorithm 5 computes a C-approximate barycentric spanner for $\{\phi(x, a) : a \in \mathcal{A}\}$ within $O(d \log_C d)$ iterations of the while-loop.*

Lemma 6. *Fix any constant $C > 1$. Algorithm 5 can be implemented with runtime $O(\mathcal{T}_{\mathsf{Opt}} \cdot d^2 \log d + d^4 \log d)$ and memory $O(\mathcal{M}_{\mathsf{Opt}} + d^2)$.*

Proof of Lemma 6. We provide the computational complexity analysis starting from the while-loop (line 5-12) in the following. The computational complexity regarding the first for-loop (line 1-3) can be similarly analyzed.

- *Outer loops (lines 5-6).* From Lemma 5, we know that Algorithm 5 terminates within $O(d \log d)$ iterations of the while-loop (line 5). It is also clear that the for-loop (line 6) is invoked at most d times.
- *Computational complexity for lines 7-10.* We discuss how to efficiently implement this part using rank-one updates. We analyze the computational complexity for each line in the following.
  - *Line 7.* We discuss how to efficiently compute the linear function θ through rank-one updates. Fix any $Y \in \mathbb{R}^d$. Let $\Phi_{\mathcal{S}}$ denote the invertible (by construction) matrix whose k-th column is $\phi(x, a_k)$ (with $a_k \in \mathcal{S}$). Using the rank-one update formula for the determinant (Meyer, 2000), we have

$$\begin{aligned}
&\det(\phi(x, a_1), \dots, \phi(x, a_{i-1}), Y, \phi(x, a_{i+1}), \dots, \phi(x, a_d)) \\
&= \det\Big(\Phi_{\mathcal{S}} + \big(Y - \phi(x, a_i)\big) e_i^\top\Big) \\
&= \det(\Phi_{\mathcal{S}}) \cdot \Big(1 + e_i^\top \Phi_{\mathcal{S}}^{-1} \big(Y - \phi(x, a_i)\big)\Big) \\
&= \big\langle Y, \det(\Phi_{\mathcal{S}}) \cdot \big(\Phi_{\mathcal{S}}^{-1}\big)^\top e_i \big\rangle + \det(\Phi_{\mathcal{S}}) \cdot \big(1 - e_i^\top \Phi_{\mathcal{S}}^{-1} \phi(x, a_i)\big). \qquad (8)
\end{aligned}$$

    We first notice that $\det(\Phi_{\mathcal{S}}) \cdot \big(1 - e_i^\top \Phi_{\mathcal{S}}^{-1} \phi(x, a_i)\big) = 0$ since one can take $Y = 0 \in \mathbb{R}^d$. We can then write

$$\det(\phi(x, a_1), \dots, \phi(x, a_{i-1}), Y, \phi(x, a_{i+1}), \dots, \phi(x, a_d)) = \langle Y, \theta \rangle$$

where $\theta = \det(\Phi_{\mathcal{S}}) \cdot \left(\Phi_{\mathcal{S}}^{-1}\right)^{\top} e_i$. Thus, whenever $\det(\Phi_{\mathcal{S}})$ and $\Phi_{\mathcal{S}}^{-1}$ are known, compute θ takes $O(d)$ time. The maximum memory requirement is $O(d^2)$, following from the storage of $\Phi_{\mathcal{S}}^{-1}$.

- *Line 8.* When θ is computed, we can compute a by first compute $a_+ := \arg\max_{a \in \mathcal{A}} \langle \phi(x,a), \theta \rangle$ and $a_- := \arg\max_{a \in \mathcal{A}} -\langle \phi(x,a), \theta \rangle$ and then compare the two. This process takes two oracle calls to $\mathbf{Alg}_{\mathsf{Opt}}$, which takes $O(\mathcal{T}_{\mathsf{Opt}})$ time. The maximum memory requirement is $O(\mathcal{M}_{\mathsf{Opt}} + d)$, following from the memory requirement of $\mathbf{Alg}_{\mathsf{Opt}}$ and the storage of θ.
- *Line 9.* Once θ and $\det(\Phi_{\mathcal{S}})$ are computed, checking the updating criteria takes $O(d)$ time. The maximum memory requirement is $O(d)$, following from the storage of $\phi(x,a)$ and θ.
- *Line 10.* We discuss how to efficiently update $\det(\Phi_{\mathcal{S}})$ and $\Phi_{\mathcal{S}}^{-1}$ through rank-one updates. If an update $a_i = a$ is made, we can update the determinant using rank-one update (as in Eq. (8)) with runtime $O(d)$ and memory $O(d^2)$; and update the inverse matrix using the Sherman-Morrison rank-one update formula (Sherman & Morrison, 1950), i.e.,

$$\left(\Phi_{\mathcal{S}} + \big(\phi(x,a) - \phi(x,a_i)\big) e_i^{\top}\right)^{-1} = \Phi_{\mathcal{S}}^{-1} - \frac{\Phi_{\mathcal{S}}^{-1}\big(\phi(x,a) - \phi(x,a_i)\big) e_i^{\top} \Phi_{\mathcal{S}}^{-1}}{1 + e_i \Phi_{\mathcal{S}}^{-1}\big(\phi(x,a) - \phi(x,a_i)\big)},$$

  which can be implemented in $O(d^2)$ time and memory. Note that the updated matrix must be invertible by construction.

Thus, using rank-one updates, the total runtime adds up to $O(\mathcal{T}_{\mathsf{Opt}} + d^2)$ and the maximum memory requirement is $O(\mathcal{M}_{\mathsf{Opt}} + d^2)$. We also remark that the initial matrix determinant and inverse can be computed cheaply since the first iteration of the first for-loop (i.e., line 2 with $i = 1$) is updated from the identity matrix.

To summarize, Algorithm 5 has runtime $O(\mathcal{T}_{\mathsf{Opt}} \cdot d^2 \log d + d^4 \log d)$ and uses at most $O(\mathcal{M}_{\mathsf{Opt}} + d^2)$ units of memory.

□

The next proposition shows that a barycentric spanner implies an approximate optimal design. The result is well-known (e.g., Hazan & Karnin (2016)), but we provide a proof here for completeness.

Lemma 2. *If $\mathcal{S} = \{z_1, \dots, z_d\}$ is a C-approximate barycentric spanner for $\mathcal{Z} \subseteq \mathbb{R}^d$, then $q := \mathrm{unif}(\mathcal{S})$ is a $(C^2 \cdot d)$-approximate optimal design.*

Proof of Lemma 2. Assume without loss of generality that $\mathcal{Z} \subseteq \mathbb{R}^d$ spans $\mathbb{R}^d$. By Definition 3, we know that for any $z \in \mathcal{Z}$, we can represent z as a weighted sum of elements in $\mathcal{S}$ with coefficients in the range $[-C, C]$. Let $\Phi_{\mathcal{S}} \in \mathbb{R}^{d \times d}$ be the matrix whose columns are the vectors in $\mathcal{S}$. For any $z \in \mathcal{Z}$, we can find $\theta \in [-C, C]^d$ such that $z = \Phi_{\mathcal{S}} \theta$. Since $\Phi_{\mathcal{S}}$ is invertible (by construction), we can write $\theta = \Phi_{\mathcal{S}}^{-1} z$, which implies the result via

$$C^2 \cdot d \geq \|\theta\|_2^2 = \|z\|^2_{(\Phi_{\mathcal{S}} \Phi_{\mathcal{S}}^{\top})^{-1}} = \frac{1}{d} \cdot \|z\|^2_{V(q)^{-1}}.$$

□

B.1.2. Regret Decomposition

Fix any $\gamma > 0$. We consider the following meta algorithm that utilizes the online regression oracle $\mathbf{Alg}_{\mathsf{Sq}}$ defined in Assumption 2.

For $t = 1, 2, \dots, T$:

- Get context $x_t \in \mathcal{X}$ from the environment and regression function $\widehat{f}_t \in \mathrm{conv}(\mathcal{F})$ from the online regression oracle $\mathbf{Alg}_{\mathsf{Sq}}$.
- Identify the distribution $p_t \in \Delta(\mathcal{A})$ that solves the minimax problem $\mathsf{dec}_{\gamma}(\mathcal{F}; \widehat{f}_t, x_t)$ (defined in Eq. (3)) and play action $a_t \sim p_t$.
- Observe reward r_t and update regression oracle with example (x_t, a_t, r_t).

The following result bounds the contextual bandit regret for the meta algorithm described above. The result is a variant of the regret decomposition based on the Decision-Estimation Coefficient given in Foster et al. (2021b), which generalizes Foster & Rakhlin (2020). The slight differences in constant terms are due to the difference in reward range.

Lemma 7 (Foster & Rakhlin (2020); Foster et al. (2021b))**.** *Suppose that Assumption 2 holds. Then probability at least $1-\delta$, the contextual bandit regret is upper bounded as follows:*

$$\mathbf{Reg}_{\mathsf{CB}}(T) \leq \mathsf{dec}_{\gamma}(\mathcal{F}) \cdot T + 2\gamma \cdot \mathbf{Reg}_{\mathsf{Sq}}(T) + 64\gamma \cdot \log(2\delta^{-1}) + \sqrt{8T\log(2\delta^{-1})}.$$

In general, identifying a distribution that *exactly* solves the minimax problem corresponding to the DEC may be impractical. However, if one can identify a distribution that instead certifies an *upper bound* $\overline{\mathsf{dec}}_{\gamma}(\mathcal{F})$ on the Decision-Estimation Coefficient (in the sense that $\mathsf{dec}_{\gamma}(\mathcal{F}) \leq \overline{\mathsf{dec}}_{\gamma}(\mathcal{F})$), the regret bound in Lemma 7 continues to hold with $\mathsf{dec}_{\gamma}(\mathcal{F})$ replaced by $\overline{\mathsf{dec}}_{\gamma}(\mathcal{F})$.

B.1.3. Proof of Lemma 3

Lemma 3. *For any $\gamma \geq 1$, by choosing $\varepsilon = \sqrt{C_{\mathrm{opt}} \cdot d/4\gamma} \wedge 1$, the exploration strategy in Algorithm 1 certifies that $\mathsf{dec}_{\gamma}(\mathcal{F}) = O(\sqrt{C_{\mathrm{opt}} \cdot d/\gamma})$.*

Proof of Lemma 3. Fix a context $x \in \mathcal{X}$. In our setting, where actions are linearly structured, we can equivalently write the Decision-Estimation Coefficient $\mathsf{dec}_{\gamma}(\mathcal{F}; \widehat{f}, x)$ as

$$\mathsf{dec}_{\gamma}(\mathcal{G}; \widehat{g}, x) := \inf_{p \in \Delta(\mathcal{A})} \sup_{a^\star \in \mathcal{A}} \sup_{g^\star \in \mathcal{G}} \mathbb{E}_{a\sim p}\Big[\big\langle \phi(x,a^\star) - \phi(x,a), g^\star(x)\big\rangle - \gamma \cdot \big(\big\langle \phi(x,a), g^\star(x) - \widehat{g}(x)\big\rangle\big)^2\Big]. \tag{9}$$

Recall that within our algorithms, $\widehat{g} \in \mathrm{conv}(\mathcal{G})$ is obtained from the estimator $\widehat{f} = f_{\widehat{g}}$ output by $\mathbf{Alg}_{\mathsf{Sq}}$. We will bound the quantity in Eq. (9) uniformly for all $x \in \mathcal{X}$ and $\widehat{g} : \mathcal{X} \to \mathbb{R}^d$ with $\|\widehat{g}\| \leq 1$. Recall that we assume $\sup_{g\in\mathcal{G}, x\in\mathcal{X}} \|g(x)\| \leq 1$.

Denote $\widehat{a} := \arg\max_{a\in\mathcal{A}} \big\langle \phi(x,a), \widehat{g}(x)\big\rangle$ and $a^\star := \arg\max_{a\in\mathcal{A}} \big\langle \phi(x,a), g^\star(x)\big\rangle$. For any $\varepsilon \leq 1$, let $p := \varepsilon \cdot q^{\mathrm{opt}} + (1-\varepsilon)\cdot \mathbb{I}_{\widehat{a}}$, where $q^{\mathrm{opt}} \in \Delta(\mathcal{A})$ is any C_{opt}-approximate optimal design for the embedding $\{\phi(x,a)\}_{a\in\mathcal{A}}$. We have the following decomposition.

$$\begin{aligned}\mathbb{E}_{a\sim p}\Big[\big\langle \phi(x,a^\star) - \phi(x,a), g^\star(x)\big\rangle\Big] &= \mathbb{E}_{a\sim p}\Big[\big\langle \phi(x,\widehat{a}) - \phi(x,a), \widehat{g}(x)\big\rangle\Big] + \mathbb{E}_{a\sim p}\Big[\big\langle \phi(x,a), \widehat{g}(x) - g^\star(x)\big\rangle\Big] \\ &\quad + \Big(\big\langle \phi(x,a^\star), g^\star(x)\big\rangle - \big\langle \phi(x,\widehat{a}), \widehat{g}(x)\big\rangle\Big). \end{aligned} \tag{10}$$

For the first term in Eq. (10), we have

$$\mathbb{E}_{a\sim p}\Big[\big\langle \phi(x,\widehat{a}) - \phi(x,a), \widehat{g}(x)\big\rangle\Big] = \varepsilon \cdot \mathbb{E}_{a\sim q^{\mathrm{opt}}}\Big[\big\langle \phi(x,\widehat{a}) - \phi(x,a), \widehat{g}(x)\big\rangle\Big] \leq 2\varepsilon \cdot \sup_{x\in\mathcal{X}, a\in\mathcal{A}} \|\phi(x,a)\| \cdot \sup_{x\in\mathcal{X}} \|\widehat{g}(x)\| \leq 2\varepsilon.$$

Next, since

$$\big\langle \phi(x,a), \widehat{g}(x) - g^\star(x)\big\rangle \leq \frac{\gamma}{2} \cdot \big(\big\langle \phi(x,a), \widehat{g}(x) - g^\star(x)\big\rangle\big)^2 + \frac{1}{2\gamma}$$

by AM-GM inequality, we can bound the second term in Eq. (10) by

$$\mathbb{E}_{a\sim p}\Big[\big\langle \phi(x,a), \widehat{g}(x) - g^\star(x)\big\rangle\Big] \leq \frac{\gamma}{2} \cdot \mathbb{E}_{a\sim p}\Big[\big(\big\langle \phi(x,a), \widehat{g}(x) - g^\star(x)\big\rangle\big)^2\Big] + \frac{1}{2\gamma}.$$

We now turn our attention to the third term. Observe that since $\widehat{a}$ is optimal for $\widehat{g}$, $\big\langle \phi(x,\widehat{a}), \widehat{g}(x)\big\rangle \geq \big\langle \phi(x,a^\star), \widehat{g}(x)\big\rangle$. As a result, defining $V(q^{\mathrm{opt}}) := \mathbb{E}_{a\sim q^{\mathrm{opt}}}[\phi(x,a)\phi(x,a)^\top]$, we have

$$\begin{aligned}\big\langle \phi(x,a^\star), g^\star(x)\big\rangle - \big\langle \phi(x,\widehat{a}), \widehat{g}(x)\big\rangle &\leq \big\langle \phi(x,a^\star), g^\star(x) - \widehat{g}(x)\big\rangle \\ &\leq \big\|\phi(x,a^\star)\big\|_{V(q^{\mathrm{opt}})^{-1}} \cdot \big\|g^\star(x) - \widehat{g}(x)\big\|_{V(q^{\mathrm{opt}})} \\ &= \frac{1}{2\gamma\varepsilon} \cdot \big\|\phi(x,a^\star)\big\|^2_{V(q^{\mathrm{opt}})^{-1}} + \frac{\gamma}{2} \cdot \varepsilon \cdot \mathbb{E}_{a\sim q^{\mathrm{opt}}}\Big[\big(\phi(x,a), g^\star(x) - \widehat{g}(x)\big)^2\Big] \\ &\leq \frac{C_{\mathrm{opt}} \cdot d}{2\gamma\varepsilon} + \frac{\gamma}{2} \cdot \mathbb{E}_{a\sim p}\Big[\big(\phi(x,a), g^\star(x) - \widehat{g}(x)\big)^2\Big].\end{aligned}$$

Here, the third line follows from the AM-GM inequality, and the last line follows from the (C_{opt}-approximate) optimal design property and the definition of p.

Combining these bounds, we have

$$\mathsf{dec}_\gamma(\mathcal{F}) = \inf_{p\in\Delta(\mathcal{A})} \sup_{a^\star\in\mathcal{A}} \sup_{g^\star\in\mathcal{G}} \mathsf{dec}_\gamma(\mathcal{G};\widehat{g},x) \le 2\varepsilon + \frac{1}{2\gamma} + \frac{C_{\mathrm{opt}}\cdot d}{2\gamma\varepsilon}.$$

Since $\gamma \ge 1$, taking $\varepsilon := \sqrt{C_{\mathrm{opt}}\cdot d/4\gamma} \wedge 1$ gives

$$\mathsf{dec}_\gamma(\mathcal{F}) \le 2\sqrt{\frac{C_{\mathrm{opt}}\cdot d}{\gamma}} + \frac{1}{2\gamma} \le 3\sqrt{\frac{C_{\mathrm{opt}}\cdot d}{\gamma}}$$

whenever $\varepsilon < 1$. On the other hand, when $\varepsilon = 1$, this bound holds trivially. □

B.2. Proof of Theorem 1

Theorem 1. *With a C_{opt}-approximate optimal design subroutine and an appropriate choice for $\varepsilon \in (0,1]$, Algorithm 1, with probability at least $1-\delta$, enjoys regret*

$$\begin{aligned}&\mathbf{Reg}_{\mathsf{CB}}(T)\\ &= O\Big((C_{\mathrm{opt}}\cdot d)^{1/3}T^{2/3}(\mathbf{Reg}_{\mathsf{Sq}}(T)+\log(\delta^{-1}))^{1/3}\Big).\end{aligned}$$

In particular, when invoked with Algorithm 5 (with $C=2$) as a subroutine, the algorithm enjoys regret

$$\mathbf{Reg}_{\mathsf{CB}}(T) = O\Big(d^{2/3}T^{2/3}(\mathbf{Reg}_{\mathsf{Sq}}(T)+\log(\delta^{-1}))^{1/3}\Big).$$

and has per-round runtime $O(\mathcal{T}_{\mathsf{Sq}} + \mathcal{T}_{\mathsf{Opt}}\cdot d^2\log d + d^4\log d)$ and maximum memory $O(\mathcal{M}_{\mathsf{Sq}} + \mathcal{M}_{\mathsf{Opt}} + d^2)$.

Proof of Theorem 1. Combining Lemma 3 with Lemma 7, we have

$$\mathbf{Reg}_{\mathsf{CB}}(T) \le 3T\cdot\sqrt{\frac{C_{\mathrm{opt}}\cdot d}{\gamma}} + 2\gamma\cdot\mathbf{Reg}_{\mathsf{Sq}}(T) + 64\gamma\cdot\log(2\delta^{-1}) + \sqrt{8T\log(2\delta^{-1})}.$$

The regret bound in Theorem 1 immediately follows by choosing

$$\gamma = \left(\frac{3T\sqrt{C_{\mathrm{opt}}\cdot d}}{2\mathbf{Reg}_{\mathsf{Sq}}(T)+64\log(2\delta^{-1})}\right)^{2/3}.$$

In particular, when Algorithm 5 is invoked as a subroutine with parameter $C=2$, Lemma 2 implies that we may take $C_{\mathrm{opt}} \le 4d$.

Computational complexity. We now bound the per-round computational complexity of Algorithm 1 when Algorithm 5 is used as a subroutine to compute the approximate optimal design. Outside of the call to Algorithm 5, Algorithm 1 uses $O(1)$ calls to $\mathbf{Alg}_{\mathsf{Sq}}$ to obtain $\widehat{g}_t(x_t)\in\mathbb{R}^d$ and to update $\widehat{f}_t$, and uses a single call to $\mathbf{Alg}_{\mathsf{Opt}}$ to compute $\widehat{a}_t$. With the optimal design q_t^{opt} returned by Algorithm 5 (represented as a barycentric spanner), sampling from p_t takes at most $O(d)$ time, since $|\mathrm{supp}(p_t)| \le d+1$. outside of Algorithm 5 adds up to $O(\mathcal{T}_{\mathsf{Sq}} + \mathcal{T}_{\mathsf{Opt}} + d)$. In terms of memory, calling $\mathbf{Alg}_{\mathsf{Sq}}$ and $\mathbf{Alg}_{\mathsf{Opt}}$ takes $O(\mathcal{M}_{\mathsf{Sq}} + \mathcal{M}_{\mathsf{Opt}})$ units, and maintaining the distribution p_t (the barycentric spanner) takes $O(d)$ units, so the maximum memory (outside of Algorithm 5) is $O(\mathcal{M}_{\mathsf{Sq}} + \mathcal{M}_{\mathsf{Opt}} + d)$. The stated results follow from combining the computational complexities analyzed in Lemma 6.

□

C. Proofs and Supporting Results from Section 4.1

In this section we provide supporting results concerning Algorithm 2 (Appendix C.1), and then give the proof of Theorem 2 (Appendix C.2).

C.1. Supporting Results

Lemma 8. *In Algorithm 2 (Eq. (6)), there exists a unique choice of $\lambda > 0$ such that $\sum_{a\in\mathcal{A}} p_t(a) = 1$, and its value lies in $[\frac{1}{2}, 1]$.*

Proof of Lemma 8. Define $h(\lambda) := \sum_{a\in\mathrm{supp}(q_t)} \frac{q_t(a)}{\lambda+\eta(\widehat{f}_t(x_t,\widehat{a}_t)-\widehat{f}_t(x_t,a))}$. We first notice that $h(\lambda)$ is continuous and strictly decreasing over $(0,\infty)$. We further have

$$h(1/2) \geq \frac{q_t(\widehat{a}_t)}{1/2+\eta(\widehat{f}_t(x_t,\widehat{a}_t)-\widehat{f}_t(x_t,\widehat{a}_t))} \geq \frac{1/2}{1/2} = 1;$$

and

$$h(1) \leq \sum_{a\in\mathrm{supp}(q_t)} q_t(a) = \frac{1}{2}+\frac{1}{2}\sum_{a\in\mathrm{supp}(q_t^{\mathrm{opt}})} q_t^{\mathrm{opt}}(a) = 1.$$

As a result, there exists a unique normalization constant $\lambda^\star \in [\frac{1}{2}, 1]$ such that $h(\lambda^\star) = 1$. □

Lemma 4. *For any $\gamma > 0$, by setting $\eta = \gamma/(C_{\mathrm{opt}}\cdot d)$, the exploration strategy used in Algorithm 2 certifies that $\mathsf{dec}_\gamma(\mathcal{F}) = O(C_{\mathrm{opt}}\cdot d/\gamma)$.*

Proof of Lemma 4. As in the proof of Lemma 3, we use the linear structure of the action space to rewrite the Decision-Estimation Coefficient $\mathsf{dec}_\gamma(\mathcal{F};\widehat{f},x)$ as

$$\mathsf{dec}_\gamma(\mathcal{G};\widehat{g},x) := \inf_{p\in\Delta(\mathcal{A})}\sup_{a^\star\in\mathcal{A}}\sup_{g^\star\in\mathcal{G}} \mathbb{E}_{a\sim p}\Big[\big\langle\phi(x,a^\star)-\phi(x,a),g^\star(x)\big\rangle - \gamma\cdot\big(\big\langle\phi(x,a),g^\star(x)-\widehat{g}(x)\big\rangle\big)^2\Big],$$

Where $\widehat{g}$ is such that $\widehat{f} = f_{\widehat{g}}$. We will bound the quantity above uniformly for all $x\in\mathcal{X}$ and $\widehat{g}:\mathcal{X}\to\mathbb{R}^d$.

Denote $\widehat{a} := \arg\max_{a\in\mathcal{A}}\langle\phi(x,a),\widehat{g}(x)\rangle$, $a^\star := \arg\max_{a\in\mathcal{A}}\langle\phi(x,a),g^\star(x)\rangle$ and $q^{\mathrm{opt}}\in\Delta(\mathcal{A})$ be a C_{opt}-approximate optimal design with respect to the reweighted embedding $\bar{\phi}(x,\cdot)$). We use the setting $\eta = \frac{\gamma}{C_{\mathrm{opt}}\cdot d}$ throughout the proof. Recall that for the sampling distribution in Algorithm 2, we set $q := \frac{1}{2}q^{\mathrm{opt}} + \frac{1}{2}\mathbb{I}_{\widehat{a}}$ and define

$$p(a) = \frac{q(a)}{\lambda+\frac{\gamma}{C_{\mathrm{opt}}\cdot d}\big(\big\langle\phi(x,\widehat{a})-\phi(x,a),\widehat{g}(x)\big\rangle\big)}, \tag{11}$$

where $\lambda\in[\frac{1}{2},1]$ is a normalization constant (cf. Lemma 8).

We decompose the regret of the distribution p in Eq. (11) as

$$\begin{aligned}\mathbb{E}_{a\sim p}\Big[\big\langle\phi(x,a^\star)-\phi(x,a),g^\star(x)\big\rangle\Big] &= \mathbb{E}_{a\sim p}\Big[\big\langle\phi(x,\widehat{a})-\phi(x,a),\widehat{g}(x)\big\rangle\Big] + \mathbb{E}_{a\sim p}\Big[\big\langle\phi(x,a),\widehat{g}(x)-g^\star(x)\big\rangle\Big]\\ &\quad + \big\langle\phi(x,a^\star),g^\star(x)-\widehat{g}(x)\big\rangle + \big\langle\phi(x,a^\star)-\phi(x,\widehat{a}),\widehat{g}(x)\big\rangle.\end{aligned} \tag{12}$$

Writing out the expectation, the first term in Eq. (12) is upper bounded as follows.

$$\begin{aligned}\mathbb{E}_{a\sim p}\Big[\big\langle\phi(x,\widehat{a})-\phi(x,a),\widehat{g}(x)\big\rangle\Big] &= \sum_{a\in\mathrm{supp}(q^{\mathrm{opt}})\cup\{\widehat{a}\}} p(a)\cdot\big\langle\phi(x,\widehat{a})-\phi(x,a),\widehat{g}(x)\big\rangle\\ &< \sum_{a\in\mathrm{supp}(q^{\mathrm{opt}})}\frac{q^{\mathrm{opt}}(a)/2}{\frac{\gamma}{C_{\mathrm{opt}}\cdot d}\big(\big\langle\phi(x,\widehat{a})-\phi(x,a),\widehat{g}(x)\big\rangle\big)}\cdot\big\langle\phi(x,\widehat{a})-\phi(x,a),\widehat{g}(x)\big\rangle\\ &\leq \frac{C_{\mathrm{opt}}\cdot d}{2\gamma},\end{aligned}$$

where we use that $\lambda > 0$ in the second inequality (with the convention that $\frac{0}{0} = 0$).

The second term in Eq. (12) can be upper bounded as in the proof of Lemma 3, by applying the AM-GM inequality:

$$\mathbb{E}_{a\sim p}\Big[\big\langle\phi(x,a),g^\star(x)-\widehat{g}(x)\big\rangle\Big]\leq\frac{\gamma}{2}\cdot\mathbb{E}_{a\sim p}\Big[\big(\big\langle\phi(x,a),\widehat{g}(x)-g^\star(x)\big\rangle\big)^2\Big]+\frac{1}{2\gamma}.$$

The third term in Eq. (12) is the most involved. To begin, we define $V(p):=\mathbb{E}_{a\sim p}[\phi(x,a)\phi(x,a)^\top]$ and apply the following standard bound:

$$\begin{aligned}\big\langle\phi(x,a^\star),\widehat{g}(x)-g^\star(x)\big\rangle&\leq\big\|\phi(x,a^\star)\big\|_{V(p)^{-1}}\cdot\big\|g^\star(x)-\widehat{g}(x)\big\|_{V(p)}\\&\leq\frac{1}{2\gamma}\cdot\big\|\phi(x,a^\star)\big\|^2_{V(p)^{-1}}+\frac{\gamma}{2}\cdot\big\|g^\star(x)-\widehat{g}(x)\big\|^2_{V(p)}\\&=\frac{1}{2\gamma}\cdot\big\|\phi(x,a^\star)\big\|^2_{V(p)^{-1}}+\frac{\gamma}{2}\cdot\mathbb{E}_{a\sim p}\Big[\big(\phi(x,a),g^\star(x)-\widehat{g}(x)\big)^2\Big],\end{aligned}\tag{13}$$

where the second line follows from the AM-GM inequality. The second term in Eq. (13) matches the bound we desired, so it remains to bound the first term. Let $\check{q}^{\mathrm{opt}}$ be the following sub-probability measure:

$$\check{q}^{\mathrm{opt}}(a):=\frac{q^{\mathrm{opt}}(a)/2}{\lambda+\frac{\gamma}{C_{\mathrm{opt}}\cdot d}\big(\big\langle\phi(x,\widehat{a})-\phi(x,a),\widehat{g}(x)\big\rangle\big)},$$

and let $V(\check{q}^{\mathrm{opt}}):=\mathbb{E}_{a\sim\check{q}^{\mathrm{opt}}}[\phi(x,a)\phi(x,a)^\top]$. We clearly have $V(p)\succeq V(\check{q}^{\mathrm{opt}})$ from the definition of p (cf. Eq. (11)). We observe that

$$\begin{aligned}V(\check{q}^{\mathrm{opt}})&=\sum_{a\in\mathrm{supp}(\check{q}^{\mathrm{opt}})}\check{q}^{\mathrm{opt}}(a)\phi(x,a)\phi(x,a)^\top\\&=\frac{1}{2}\cdot\sum_{a\in\mathrm{supp}(q^{\mathrm{opt}})}q^{\mathrm{opt}}(a)\bar{\phi}(x,a)\bar{\phi}(x,a)^\top\cdot\frac{1+\frac{\gamma}{C_{\mathrm{opt}}\cdot d}\big(\big\langle\phi(x,\widehat{a})-\phi(x,a),\widehat{g}(x)\big\rangle\big)}{\lambda+\frac{\gamma}{C_{\mathrm{opt}}\cdot d}\big(\big\langle\phi(x,\widehat{a})-\phi(x,a),\widehat{g}(x)\big\rangle\big)}\\&\succeq\frac{1}{2}\cdot\sum_{a\in\mathrm{supp}(q^{\mathrm{opt}})}q^{\mathrm{opt}}(a)\bar{\phi}(x,a)\bar{\phi}(x,a)^\top=:\frac{1}{2}\overline{V}(q^{\mathrm{opt}}),\end{aligned}$$

where the last line uses that $\lambda\leq 1$. Since $\overline{V}(q^{\mathrm{opt}})$ is positive-definite by construction, we have that $V(p)^{-1}\preceq V(\check{q}^{\mathrm{opt}})^{-1}\preceq 2\cdot\overline{V}(q^{\mathrm{opt}})^{-1}$. As a result,

$$\begin{aligned}\frac{1}{2\gamma}\cdot\big\|\phi(x,a^\star)\big\|^2_{V(p)^{-1}}&\leq\frac{1}{\gamma}\cdot\big\|\phi(x,a^\star)\big\|^2_{\overline{V}(q^{\mathrm{opt}})^{-1}}\\&=\frac{1+\frac{\gamma}{C_{\mathrm{opt}}\cdot d}\big(\big\langle\phi(x,\widehat{a})-\phi(x,a^\star),\widehat{g}(x)\big\rangle\big)}{\gamma}\cdot\big\|\bar{\phi}(x,a^\star)\big\|^2_{\overline{V}(q^{\mathrm{opt}})^{-1}}\\&\leq\frac{C_{\mathrm{opt}}\cdot d}{\gamma}+\big\langle\phi(x,\widehat{a})-\phi(x,a^\star),\widehat{g}(x)\big\rangle,\end{aligned}\tag{14}$$

where the last line uses that $\big\|\bar{\phi}(x,a^\star)\big\|^2_{\overline{V}(q^{\mathrm{opt}})^{-1}}\leq C_{\mathrm{opt}}\cdot d$, since q^{opt} is a C_{opt}-approximate optimal design for the set $\big\{\bar{\phi}(x,a)\big\}_{a\in\mathcal{A}}$. Finally, we observe that the second term in Eq. (14) is cancelled out by the forth term in Eq. (12).

Summarizing the bounds on the terms in Eq. (12) leads to:

$$\mathsf{dec}_\gamma(\mathcal{F})=\inf_{p\in\Delta(\mathcal{A})}\sup_{a^\star\in\mathcal{A}}\sup_{g^\star\in\mathcal{G}}\mathsf{dec}_\gamma(\mathcal{G};\widehat{g},x)\leq\frac{C_{\mathrm{opt}}\cdot d}{2\gamma}+\frac{1}{2\gamma}+\frac{C_{\mathrm{opt}}\cdot d}{\gamma}\leq\frac{2\,C_{\mathrm{opt}}\cdot d}{\gamma}.$$

□

C.2. Proof of Theorem 2

Theorem 2. *Let $\delta\in(0,1)$ be given. With a C_{opt}-approximate optimal design subroutine and an appropriate choice for $\gamma>0$, Algorithm 2 ensures that with probability at least $1-\delta$,*

$$\mathbf{Reg}_{\mathsf{CB}}(T)=O\bigg(\sqrt{C_{\mathrm{opt}}\cdot d\,T\,\big(\mathbf{Reg}_{\mathsf{Sq}}(T)+\log(\delta^{-1})\big)}\bigg).$$

In particular, when invoked with Algorithm 3 (with $C = 2$) as a subroutine, the algorithm has

$$\mathbf{Reg}_{\mathsf{CB}}(T) = O\left(d\sqrt{T\left(\mathbf{Reg}_{\mathsf{Sq}}(T) + \log(\delta^{-1})\right)}\right),$$

and has per-round runtime $O(\mathcal{T}_{\mathsf{Sq}} + (\mathcal{T}_{\mathsf{Opt}} \cdot d^3 + d^4) \cdot \log^2\left(\frac{T}{r}\right))$ and the maximum memory $O(\mathcal{M}_{\mathsf{Sq}} + \mathcal{M}_{\mathsf{Opt}} + d^2 + d\log\left(\frac{T}{r}\right))$.

Proof. Combining Lemma 4 with Lemma 7, we have

$$\mathbf{Reg}_{\mathsf{CB}}(T) \leq 2T \cdot \frac{C_{\mathrm{opt}} \cdot d}{\gamma} + 2\gamma \cdot \mathbf{Reg}_{\mathsf{Sq}}(T) + 64\gamma \cdot \log(2\delta^{-1}) + \sqrt{8T\log(2\delta^{-1})}.$$

The theorem follows by choosing

$$\gamma = \left(\frac{C_{\mathrm{opt}} \cdot d\,T}{\mathbf{Reg}_{\mathsf{Sq}}(T) + 32\log(2\delta^{-1})}\right)^{1/2}.$$

In particular, when Algorithm 3 is invoked as the subroutine with parameter $C = 2$, we may take $C_{\mathrm{opt}} = 4d$.

Computational complexity. We now discuss the per-round computational complexity of Algorithm 2. We analyze a variant of the sampling rule specified in Appendix E.2 that does not require computation of the normalization constant. Outside of the runtime and memory requirements required to compute the barycentric spanner using Algorithm 3, which are stated in Theorem 3, Algorithm 2 uses $O(1)$ calls to the oracle $\mathbf{Alg}_{\mathsf{Sq}}$ to obtain $\widehat{g}_t(x_t) \in \mathbb{R}^d$ and update $\widehat{f}_t$, and uses a single call to $\mathbf{Alg}_{\mathsf{Opt}}$ to compute $\widehat{a}_t$. With $\widehat{g}_t(x_t)$ and $\widehat{a}_t$, we can compute $\widehat{f}_t(x_t, \widehat{a}_t) - \widehat{f}_t(x_t, a) = \langle \phi(x_t, \widehat{a}_t) - \phi(x_t, a), \widehat{g}_t(x_t)\rangle$ in $O(d)$ time for any $a \in \mathcal{A}$; thus, with the optimal design q_t^{opt} returned by Algorithm 3 (represented as a barycentric spanner), we can construct the sampling distribution p_t in $O(d^2)$ time. Sampling from p_t takes $O(d)$ time since $|\mathrm{supp}(p_t)| \leq d + 1$. This adds up to runtime $O(\mathcal{T}_{\mathsf{Sq}} + \mathcal{T}_{\mathsf{Opt}} + d^2)$. In terms of memory, calling $\mathbf{Alg}_{\mathsf{Sq}}$ and $\mathbf{Alg}_{\mathsf{Opt}}$ takes $O(\mathcal{M}_{\mathsf{Sq}} + \mathcal{M}_{\mathsf{Opt}})$ units, and maintaining the distribution p_t (the barycentric spanner) takes $O(d)$ units, so the maximum memory (outside of Algorithm 3) is $O(\mathcal{M}_{\mathsf{Sq}} + \mathcal{M}_{\mathsf{Opt}} + d)$. The stated results follow from combining the computational complexities analyzed in Theorem 3 , together with the choice of γ described above. □

D. Proofs and Supporting Results from Section 4.2

This section of the appendix is dedicated to the analysis of Algorithm 3, and organized as follows.

- First, in Appendix D.1, we analyze Algorithm 4, a subroutine of Algorithm 3 which implements a linear optimization oracle for the reweighted action set used in the algorithm.
- Next, in Appendix D.2, we prove Theorem 3, the main theorem concerning the performance of Algorithm 3.
- Finally, in Appendix D.3, we discuss settings in which the initialization step required by Algorithm 3 can be performed efficiently.

Throughout this section of the appendix, we assume that the context $x \in \mathcal{X}$ and estimator $\widehat{g} : \mathcal{X} \to \mathbb{R}^d$—which are arguments to Algorithm 3 and Algorithm 4—are fixed.

D.1. Analysis of Algorithm 4 (Linear Optimization Oracle for Reweighted Embeddings)

A first step is to construct an (approximate) argmax oracle (after taking absolute value) with respect to the reweighted embedding $\bar{\phi}$. Recall that the goal of Algorithm 4 is to implement a linear optimization oracle for the reweighted embeddings constructed by Algorithm 3. That is, for any $\theta \in \mathbb{R}^d$, we would like to compute an action that (approximately) solves

$$\arg\max_{a\in\mathcal{A}} \left|\left\langle \bar{\phi}(x,a), \theta\right\rangle\right| = \arg\max_{a\in\mathcal{A}} \left\langle \bar{\phi}(x,a), \theta\right\rangle^2.$$

Define

$$\iota(a) := \langle \bar{\phi}(x,a), \theta\rangle^2, \quad \text{and} \quad a^\star := \arg\max_{a\in\mathcal{A}} \iota(a). \tag{15}$$

The main result of this section, Theorem 4, shows that Algorithm 4 identifies an action that achieves the maximum value in Eq. (15) up to a multiplicative constant.

Theorem 4. *Fix any $\eta > 0$, $r \in (0,1)$. Suppose $\zeta \leq \sqrt{\iota(a^\star)} \leq 1$ for some $\zeta > 0$. Then Algorithm 4 identifies an action $\breve{a}$ such that $\sqrt{\iota(\breve{a})} \geq \frac{\sqrt{2}}{2} \cdot \sqrt{\iota(a^\star)}$, and does so with runtime $O((\mathcal{T}_{\mathsf{Opt}} + d) \cdot \log(e \vee \frac{\eta}{\zeta}))$ and maximum memory $O(\mathcal{M}_{\mathsf{Opt}} + \log(e \vee \frac{\eta}{\zeta}) + d)$.*

Proof of Theorem 4. Recall from Eq. (5) that we have

$$\langle \bar{\phi}(x,a), \theta \rangle^2 = \left(\frac{\langle \phi(x,a), \theta \rangle}{\sqrt{1 + \eta \langle \phi(x,\widehat{a}) - \phi(x,a), \widehat{g}(x) \rangle}} \right)^2 = \frac{\langle \phi(x,a), \theta \rangle^2}{1 + \eta \langle \phi(x,\widehat{a}) - \phi(x,a), \widehat{g}(x) \rangle},$$

where $\widehat{a} := \arg\max_{a \in \mathcal{A}} \langle \phi(x,a), \widehat{g}(x) \rangle$; note that the denominator is at least 1. To proceed, we use that for any $X \in \mathbb{R}$ and $Y^2 > 0$, we have

$$\frac{X^2}{Y^2} = \sup_{\varepsilon \in \mathbb{R}} \{ 2\varepsilon X - \varepsilon^2 Y^2 \}.$$

Taking $X = \langle \phi(x,a), \theta \rangle$ and $Y^2 = 1 + \eta \langle \phi(x,\widehat{a}) - \phi(x,a), \widehat{g}(x) \rangle$ above, we can write

$$\begin{aligned}
\langle \bar{\phi}(x,a), \theta \rangle^2 &= \frac{\langle \phi(x,a), \theta \rangle^2}{1 + \eta \langle \phi(x,\widehat{a}) - \phi(x,a), \widehat{g}(x) \rangle} \\
&= \sup_{\varepsilon \in \mathbb{R}} \Big\{ 2\varepsilon \langle \phi(x,a), \theta \rangle - \varepsilon^2 \cdot \big(1 + \eta \langle \phi(x,\widehat{a}) - \phi(x,a), \widehat{g}(x) \rangle\big) \Big\} \qquad (16) \\
&= \sup_{\varepsilon \in \mathbb{R}} \Big\{ \langle \phi(x,a), 2\varepsilon\theta + \eta\varepsilon^2 \widehat{g}(x) \rangle - \varepsilon^2 - \eta\varepsilon^2 \langle \phi(x,\widehat{a}), \widehat{g}(x) \rangle \Big\}. \qquad (17)
\end{aligned}$$

The key property of this representation is that for any *fixed* $\varepsilon \in \mathbb{R}$, Eq. (17) is a linear function of the *unweighted* embedding ϕ, and hence can be optimized using $\mathbf{Alg}_{\mathsf{Opt}}$. In particular, for any fixed $\varepsilon \in \mathbb{R}$, consider the following linear optimization problem, which can be solved by calling $\mathbf{Alg}_{\mathsf{Opt}}$:

$$\arg\max_{a \in \mathcal{A}} \Big\{ 2\varepsilon \langle \phi(x,a), \theta \rangle - \varepsilon^2 \cdot \big(1 + \eta \langle \phi(x,\widehat{a}) - \phi(x,a), \widehat{g}(x) \rangle\big) \Big\} =: \arg\max_{a \in \mathcal{A}} W(a;\varepsilon). \qquad (18)$$

Define

$$\varepsilon^\star := \frac{\langle \phi(x,a^\star), \theta \rangle}{1 + \eta \langle \phi(x,\widehat{a}) - \phi(x,a^\star), \widehat{g}(x) \rangle}. \qquad (19)$$

If $\varepsilon^\star$ was known (which is not the case, since $a^\star$ is unknown), we could set $\varepsilon = \varepsilon^\star$ in Eq. (18) and compute an action $\bar{a} := \arg\max_{a \in \mathcal{A}} W(a;\varepsilon^\star)$ using a single oracle call. We would then have $\iota(\bar{a}) \geq W(\bar{a};\varepsilon^\star) \geq W(a^\star;\varepsilon^\star) = \iota(a^\star)$, which follows because $\varepsilon^\star$ is the maximizer in Eq. (16) for $a = a^\star$.

To get around the fact that $\varepsilon^\star$ is unknown, Algorithm 4 performs a grid search over possible values of ε. To show that the procedure succeeds, we begin by bounding the range of $\varepsilon^\star$. With some rewriting, we have

$$|\varepsilon^\star| = \frac{\sqrt{\iota(a^\star)}}{\sqrt{1 + \eta \langle \phi(x,\widehat{a}) - \phi(x,a^\star), \widehat{g}(x) \rangle}}.$$

Since $0 < \zeta \leq \sqrt{\iota(a^\star)} \leq 1$, we have

$$\bar{\zeta} := \frac{\zeta}{\sqrt{1+2\eta}} \leq |\varepsilon^\star| \leq 1.$$

Algorithm 4 performs a $(3/4)$-multiplicative grid search over the intervals $[\bar{\zeta}, 1]$ and $[-1, -\bar{\zeta}]$, which uses $2\lceil \log_{\frac{4}{3}}(\bar{\zeta}^{-1}) \rceil = O(\log(e \vee \frac{\eta}{\zeta}))$ grid points. It is immediate to that the grid contains $\bar{\varepsilon} \in \mathbb{R}$ such that $\bar{\varepsilon} \cdot \varepsilon^\star > 0$ and $\frac{3}{4}|\varepsilon^\star| \leq |\bar{\varepsilon}| \leq |\varepsilon^\star|$.

Invoking Lemma 9 (stated and proven in the sequel) with $\bar{a} := \arg\max_{a\in\mathcal{A}} W(a;\bar{\varepsilon})$ implies that $\iota(\bar{a}) \geq \frac{1}{2}\iota(a^\star)$. To conclude, recall that Algorithm 4 outputs the maximizer

$$\check{a} := \arg\max_{a\in\widehat{\mathcal{A}}} \iota(a),$$

where $\widehat{\mathcal{A}}$ is the set of argmax actions encountered by the grid search. Since $\bar{a} \in \widehat{\mathcal{A}}$, we have $\iota(\check{a}) \geq \iota(\bar{a}) \geq \frac{1}{2}\iota(a^\star)$ as desired.

Computational complexity. Finally, we bound the computational complexity of Algorithm 4. Algorithm 4 maintains a grid of $O(\log(e \vee \frac{\eta}{\zeta}))$ points, and hence calls the oracle $\mathbf{Alg}_{\mathsf{Opt}}$ $O(\log(e \vee \frac{\eta}{\zeta}))$ in total; this takes $O(\mathcal{T}_{\mathsf{Opt}} \cdot \log(e \vee \frac{\eta}{\zeta}))$ time. Computing the final maximizer from the set $\widehat{\mathcal{A}}$, which contains $O(\log(e \vee \frac{\eta}{\zeta}))$ actions, takes $O(d\log(e \vee \frac{\eta}{\zeta}))$ time (compute each $\langle\bar{\phi}(x,a),\bar{\theta}\rangle^2$ takes $O(d)$ time). Hence, the total runtime of Algorithm 4 adds up to $O((\mathcal{T}_{\mathsf{Opt}} + d)\cdot\log(e \vee \frac{\eta}{\zeta}))$. The maximum memory requirement is $O(\mathcal{M}_{\mathsf{Opt}} + \log(e \vee \frac{\eta}{\zeta}) + d)$, follows from calling $\mathbf{Alg}_{\mathsf{Opt}}$, and storing $\mathcal{E}, \widehat{\mathcal{A}}$ and other terms such as $\widehat{g}(x), \theta, \bar{\theta}, \phi(x,a), \bar{\phi}(x,a)$. □

D.1.1. Supporting Results

Lemma 9. *Let $\varepsilon^\star$ be defined as in Eq. (19). Suppose $\bar{\varepsilon} \in \mathbb{R}$ has $\bar{\varepsilon}\cdot\varepsilon^\star > 0$ and $\frac{3}{4}|\varepsilon^\star| \leq |\bar{\varepsilon}| \leq |\varepsilon^\star|$. Then, if $\bar{a} := \arg\max_{a\in\mathcal{A}} W(a;\bar{\varepsilon})$, we have $\iota(\bar{a}) \geq \frac{1}{2}\iota(a^\star)$.*

Proof of Lemma 9. First observe that using the definition of $\iota(a)$, along with Eq. (16) and Eq. (18), we have $\iota(\bar{a}) \geq W(\bar{a};\bar{\varepsilon}) \geq W(a^\star;\bar{\varepsilon})$, where the second inequality uses that $\bar{a} := \arg\max_{a\in\mathcal{A}} W(a;\bar{\varepsilon})$. Since $\bar{\varepsilon}\cdot\varepsilon^\star > 0$, we have $\mathrm{sign}(\bar{\varepsilon}\cdot\langle\phi(x,a^\star),\theta\rangle) = \mathrm{sign}(\varepsilon^\star\cdot\langle\phi(x,a^\star),\theta\rangle)$. If $\mathrm{sign}(\bar{\varepsilon}\cdot\langle\phi(x,a^\star),\theta\rangle) \geq 0$, then since $\frac{3}{4}|\varepsilon^\star| \leq |\bar{\varepsilon}| \leq |\varepsilon^\star|$, we have

$$\begin{aligned}
W(a^\star;\bar{\varepsilon}) &= 2\bar{\varepsilon}\cdot\big\langle\phi(x,a^\star),\theta\big\rangle - \bar{\varepsilon}^2\cdot\big(1+\eta\big\langle\phi(x,\widehat{a})-\phi(x,a^\star),\widehat{g}(x)\big\rangle\big)\\
&\geq \frac{3}{2}\varepsilon^\star\cdot\big\langle\phi(x,a^\star),\theta\big\rangle - (\varepsilon^\star)^2\cdot\big(1+\eta\big\langle\phi(x,\widehat{a})-\phi(x,a^\star),\widehat{g}(x)\big\rangle\big)\\
&= \frac{1}{2}\frac{\langle\phi(x,a^\star),\theta\rangle^2}{1+\eta\langle\phi(x,\widehat{a})-\phi(x,a^\star),\widehat{g}(x)\rangle} = \frac{1}{2}\iota(a^\star),
\end{aligned}$$

where we use that $1+\eta\big\langle\phi(x,\widehat{a})-\phi(x,a^\star),\widehat{g}(x)\big\rangle \geq 1$ for the first inequality and use the definition of $\varepsilon^\star$ for the second equality.

On the other hand, when $\mathrm{sign}(\bar{\varepsilon}\cdot\langle\phi(x,a^\star),\theta\rangle) < 0$, we similarly have

$$\begin{aligned}
W(a^\star;\bar{\varepsilon}) &= 2\bar{\varepsilon}\cdot\big\langle\phi(x,a^\star),\theta\big\rangle - \bar{\varepsilon}^2\cdot\big(1+\eta\big\langle\phi(x,\widehat{a})-\phi(x,a^\star),\widehat{g}(x)\big\rangle\big)\\
&\geq 2\varepsilon^\star\cdot\big\langle\phi(x,a^\star),\theta\big\rangle - (\varepsilon^\star)^2\cdot\big(1+\eta\big\langle\phi(x,\widehat{a})-\phi(x,a^\star),\widehat{g}(x)\big\rangle\big) = \iota(a^\star).
\end{aligned}$$

Summarizing both cases, we have $\iota(\bar{a}) \geq \frac{1}{2}\iota(a^\star)$. □

D.2. Proof of Theorem 3

Theorem 3. *Suppose that Algorithm 3 is invoked with parameters $\eta > 0$, $r \in (0,1)$, and $C > \sqrt{2}$, and that the initialization set $\mathcal{S}$ satisfies $|\det(\phi(x,\mathcal{S}))| \geq r^d$. Then the algorithm returns a C-approximate barycentric spanner with respect to the reweighted embedding set $\{\bar{\phi}(x,a)\}_{a\in\mathcal{A}}$, and does so with $O((\mathcal{T}_{\mathsf{Opt}}\cdot d^3 + d^4)\cdot\log^2(e\vee\frac{\eta}{r}))$ runtime and $O(\mathcal{M}_{\mathsf{Opt}} + d^2 + d\log(e\vee\frac{\eta}{r}))$ memory.*

Proof of Theorem 3. We begin by examining the range of $\sqrt{\iota(a^\star)}$ used in Theorem 4. Note that the linear function θ passed as an argument to Algorithm 3 takes the form $\bar{\phi}(x,a) \mapsto \det(\bar{\phi}(x,\mathcal{S}_i(a)))$, i.e., $\langle\bar{\phi}(x,a),\theta\rangle = \det(\bar{\phi}(x,\mathcal{S}_i(a)))$, where $\mathcal{S}_i(a) := (a_1,\ldots,a_{i-1},a,a_{i+1},\ldots,a_d)$. For the upper bound, we have

$$|\langle\bar{\phi}(x,a^\star),\theta\rangle| = |\det(\bar{\phi}(x,\mathcal{S}_i(a^\star)))| \leq \prod_{a\in\mathcal{S}_i(a^\star)}\|\bar{\phi}(x,a)\|_2^d \leq \sup_{a\in\mathcal{A}}\|\phi(x,a)\|_2^d \leq 1$$

by Hadamard's inequality and the fact that the reweighting appearing in Eq. (5) enjoys $\|\bar{\phi}(x,a)\|_2 \le \|\phi(x,a)\|_2$. This shows that $\sqrt{\iota(a^\star)} \le 1$. For the lower bound, we first recall that in Algorithm 3, the set $\mathcal{S}$ is initialized to have $|\det(\phi(x,\mathcal{S}))| \ge r^d$, and thus $|\det(\bar{\phi}(x,\mathcal{S}))| \ge \bar{r}^d$, where $\bar{r} := \frac{r}{\sqrt{1+2\eta}}$ accounts for the reweighting in Eq. (5). Next, we observe that as a consequence of the update rule in Algorithm 3, we are guaranteed that $|\det(\bar{\phi}(x,\mathcal{S}))| \ge \bar{r}^d$ across all rounds. Thus, whenever Algorithm 4 is invoked with the linear function θ described above, there must exist an action $a \in \mathcal{A}$ such that $|\langle \bar{\phi}(x,a), \theta\rangle| \ge \bar{r}^d$, which implies that $\sqrt{\iota(a^\star)} \ge \bar{r}^d$ and we can take $\zeta := \bar{r}^d$ in Theorem 4.

We next bound the number of iterations of the while-loop before the algorithm terminates. Let $\overline{C} := \frac{\sqrt{2}}{2} \cdot C > 1$. At each iteration (beginning from line 3) of Algorithm 3, one of two outcomes occurs:

1. We find an index $i \in [d]$ and an action $a \in \mathcal{A}$ such that $|\det(\bar{\phi}(x,\mathcal{S}_i(a)))| > \overline{C}|\det(\bar{\phi}(x,\mathcal{S}))|$, and update $a_i = a$.
2. We conclude that $\sup_{a\in\mathcal{A}} \max_{i\in[d]} |\det(\bar{\phi}(x,\mathcal{S}_i(a)))| \le C|\det(\bar{\phi}(x,\mathcal{S}))|$ and terminate the algorithm.

We observe that (i) the initial set $\mathcal{S}$ has $|\det(\bar{\phi}(x,\mathcal{S}))| \ge \bar{r}^d$ with $\bar{r} := \frac{r}{\sqrt{1+2\eta}}$ (as discussed before), (ii) $\sup_{\mathcal{S}\subseteq\mathcal{A},|\mathcal{S}|=d} |\det(\bar{\phi}(x,\mathcal{S}))| \le 1$ by Hadamard's inequality, and (iii) each update of $\mathcal{S}$ increases the (absolute) determinant by a factor of $\overline{C}$. Thus, fix any $C > \sqrt{2}$, we are guaranteed that Algorithm 3 terminates within $O(d\log(e \vee \frac{\eta}{r}))$ iterations of the while-loop.

We now discuss the correctness of Algorithm 3, i.e., when terminated, the set $\mathcal{S}$ is a C-approximate barycentric spanner with respect to the reweighted embedding $\bar{\phi}$. First, note that by Theorem 4, Algorithm 4 is guaranteed to identify an action $\check{a} \in \mathcal{A}$ such that $|\det(\bar{\phi}(x,\mathcal{S}_i(\check{a})))| > \overline{C}|\det(\bar{\phi}(x,\mathcal{S}))|$ as long as there exists an action $a^\star \in \mathcal{A}$ such that $|\det(\bar{\phi}(x,\mathcal{S}_i(a^\star)))| > C|\det(\bar{\phi}(x,\mathcal{S}))|$. As a result, by Observation 2.3 in Awerbuch & Kleinberg (2008), if no update is made and Algorithm 3 terminates, we have identified a C-approximate barycentric spanner with respect to embedding $\bar{\phi}$.

Computational complexity. We provide the computational complexity analysis for Algorithm 3 in the following. We use $\overline{\Phi}_{\mathcal{S}}$ to denote the matrix whose k-th column is $\bar{\phi}(x,a_k)$ with $a_k \in \mathcal{S}$.

- *Initialization.* We first notice that, given $\widehat{g}(x) \in \mathbb{R}^d$ and $\widehat{a} := \arg\max_{a\in\mathcal{A}} \langle \phi(x,a), \widehat{g}(x)\rangle$, it takes $O(d)$ time to compute $\bar{\phi}(x,a)$ for any $a \in \mathcal{A}$. Thus, computing $\det(\overline{\Phi}_{\mathcal{S}})$ and $\overline{\Phi}_{\mathcal{S}}^{-1}$ takes $O(d^2 + d^\omega) = O(d^\omega)$ time, where we use $O(d^\omega)$ (with $2 \le \omega \le 3$) to denote the time of computing matrix determinant/inversion. The maximum memory requirement is $O(d^2)$, following from the storage of $\{\bar{\phi}(x,a)\}_{a\in\mathcal{S}}$ and $\overline{\Phi}_{\mathcal{S}}^{-1}$.
- *Outer loops (lines 1-2).* We have already shown that Algorithm 5 terminates within $O(d\log(e \vee \frac{\eta}{r}))$ iterations of the while-loop (line 2). It is also clear that the for-loop (line 2) is invoked at most d times.
- *Computational complexity for lines 3-7.* We discuss how to efficiently implement this part using rank-one updates. We analyze the computational complexity for each line in the following. The analysis largely follows from the proof of Lemma 6.
  - *Line 3.* Using rank-one update of the matrix determinant (as discussed in the proof of Lemma 6), we have

    $$\det(\bar{\phi}(x,a_1),\dots,\bar{\phi}(x,a_{i-1}),Y,\bar{\phi}(x,a_{i+1}),\dots,\bar{\phi}(x,a_d)) = \langle Y,\theta\rangle,$$

    where $\theta = \det(\overline{\Phi}_{\mathcal{S}}) \cdot (\overline{\Phi}_{\mathcal{S}}^{-1})^\top e_i$. Thus, whenever $\det(\overline{\Phi}_{\mathcal{S}})$ and $\overline{\Phi}_{\mathcal{S}}^{-1}$ are known, compute θ takes $O(d)$ time. The maximum memory requirement is $O(d^2)$, following from the storage of $\overline{\Phi}_{\mathcal{S}}^{-1}$.
  - *Line 4.* When θ is computed, we can compute a by invoking IGW-ArgMax (Algorithm 4). As discussed in Theorem 4, this step takes runtime $O((\mathcal{T}_{\mathsf{Opt}}\cdot d + d^2)\cdot\log(e\vee\frac{\eta}{r}))$ and maximum memory $O(\mathcal{M}_{\mathsf{Opt}} + d\log(e\vee\frac{\eta}{r}) + d)$ (by taking $\zeta = \bar{r}^d$ as discussed before).
  - *Line 5.* Once θ and $\det(\overline{\Phi}_{\mathcal{S}})$ are computed, checking the updating criteria takes $O(d)$ time. The maximum memory requirement is $O(d)$, following from the storage of $\bar{\phi}(x,a)$ and θ.
  - *Line 6.* As discussed in the proof of Lemma 6, if an update $a_i = a$ is made, we can update $\det(\overline{\Phi}_{\mathcal{S}})$ and $\overline{\Phi}_{\mathcal{S}}^{-1}$ using rank-one updates with $O(d^2)$ time and memory.

  Thus, using rank-one updates, the total runtime for line 3-7 adds up to $O((\mathcal{T}_{\mathsf{Opt}}\cdot d + d^2)\cdot\log(e\vee\frac{\eta}{r}))$ and maximum memory requirement is $O(\mathcal{M}_{\mathsf{Opt}} + d^2 + d\log(e\vee\frac{\eta}{r}))$.

To summarize, Algorithm 5 has runtime $O((\mathcal{T}_{\mathsf{Opt}} \cdot d^3 + d^4) \cdot \log^2(e \vee \frac{\eta}{r}))$ and uses at most $O(\mathcal{M}_{\mathsf{Opt}} + d^2 + d\log(e \vee \frac{\eta}{r}))$ units of memory. □

D.3. Efficient Initializations for Algorithm 3

In this section we discuss specific settings in which the initialization required by Algorithm 3 can be computed efficiently. For the first result, we let $\mathsf{Ball}(0, r) := \{x \in \mathbb{R}^d \mid \|x\|_2 \leq r\}$ denote the ball of radius r in $\mathbb{R}^d$.

Example 2. *Suppose that there exists $r \in (0, 1)$ such that $\mathsf{Ball}(0, r) \subseteq \{\phi(x, a) : a \in \mathcal{A}\}$. Then by choosing $\mathcal{S} := \{re_1, \ldots, re_d\} \subseteq \mathcal{A}$, we have $|\det(\phi(\mathcal{S}))| = r^d$.*

The next example is stronger, and shows that we can efficiently compute a set with large determinant whenever such a set exists.

Example 3. *Suppose there exists a set $\mathcal{S}^\star \subseteq \mathcal{A}$ such that $|\det(\phi(\mathcal{S}^\star))| \geq \bar{r}^d$ for some $\bar{r} > 0$. Then there exists an efficient algorithm that identifies a set $\mathcal{S} \subseteq \mathcal{A}$ with $|\det(\phi(\mathcal{S}))| \geq r^d$ for $r := \frac{\bar{r}}{8d}$, and does so with runtime $O(\mathcal{T}_{\mathsf{Opt}} \cdot d^2 \log d + d^4 \log d)$ and memory $O(\mathcal{M}_{\mathsf{Opt}} + d^2)$.*

Proof for Example 3. The guarantee is achieved by running Algorithm 5 with $C = 2$. One can show that this strategy achieves the desired approximation guarantee by slightly generalizing the proof of a similar result in Mahabadi et al. (2019). In more detail, Mahabadi et al. (2019) study the problem of identifying a subset $\mathcal{S} \subseteq \mathcal{A}$ such that $|\mathcal{S}| = k$ and $\det(\Phi_{\mathcal{S}}^\top \Phi_{\mathcal{S}})$ is (approximately) maximized, where $\Phi_{\mathcal{S}} \in \mathbb{R}^{d \times |\mathcal{S}|}$ denotes the matrix whose columns are $\phi(x, a)$ for $a \in \mathcal{S}$. We consider the case when $k = d$, and make the following observations.

- We have $\det(\Phi_{\mathcal{S}}^\top \Phi_{\mathcal{S}}) = (\det(\Phi_{\mathcal{S}}))^2 = (\det(\phi(x, \mathcal{S})))^2$. Thus, maximizing $\det(\Phi_{\mathcal{S}}^\top \Phi_{\mathcal{S}})$ is equivalent to maximizing $|\det(\phi(x, \mathcal{S}))|$.
- The Local Search Algorithm provided in Mahabadi et al. (2019) (Algorithm 4.1 therein) has the same update and termination condition as Algorithm 5. As a result, one can show that the conclusion of their Lemma 4.1 also applies to Algorithm 5.

□

E. Other Details for Experiments

E.1. Basic Details

Datasets. oneshotwiki (Singh et al., 2012; Vasnetsov, 2018) is a named-entity recognition task where contexts are text phrases preceding and following the mention text, and where actions are text phrases corresponding to the concept names. We use the python package sentence transformers (Reimers & Gurevych, 2019) to separately embed the text preceding and following the reference into $\mathbb{R}^{768}$, and then concatenate, resulting in a context embedding in $\mathbb{R}^{1536}$. We embed the action (mentioned entity) text into $\mathbb{R}^{768}$ and then use SVD on the collection of embedded actions to reduce the dimensionality to $\mathbb{R}^{50}$. The reward function is an indicator function for whether the action corresponds to the actual entity mentioned. oneshotwiki-311 (resp. oneshotwiki-14031) is a subset of this dataset obtained by taking all actions with at least 2000 (resp. 200) examples.

amazon-3m (Bhatia et al., 2016) is an extreme multi-label dataset whose contexts are text phrases corresponding to the title and description of an item, and whose actions are integers corresponding to item tags. We separately embed the title and description phrases using sentence transformers, which leads to a context embedding in $\mathbb{R}^{1536}$. Following the protocol used in Sen et al. (2021), the first 50000 examples are fully supervised, and subsequent examples have bandit feedback. We use Hellinger PCA (Lebret & Collobert, 2014) on the supervised data label cooccurrences to construct the action embeddings in $\mathbb{R}^{800}$. Rewards are binary, and indicate whether a given item has the chosen tag. Actions that do not occur in the supervised portion of the dataset cannot be output by the model, but are retained for evaluation: For example, if during the bandit feedback phase, an example consists solely of tags that did not occur during the supervised phase, the algorithm will experience a reward of 0 for every feasible action on the example. For a typical seed, this results in roughly 890,000 feasible actions for the model. In the $(k = 5, r = 3)$ setup, we take the top-k actions as the greedy slate, and then independently decide whether to explore for each exploration slot (the bottom r slots). For exploration, we sample from the spanner set without replacement.

Regression functions and oracles. For bilinear models, regression functions take the form $f(x, a) = \langle \phi(a), Wx \rangle$, where W is a matrix of learned parameters. For deep models, regression functions pass the original context through 2 residual leaky ReLU layers before applying the bilinear layer, $f(x, a) = \langle \phi(a), W\bar{g}(x) \rangle$, where $\bar{g}$ is a learned two-layer neural network, and W is a matrix of learned parameters. For experiments with respect to oneshotwiki datasets, we add a learned bias term for regression functions (same for every action); for experiments with respect to the amazon-3m dataset, we additionally add an action-dependent bias term that is obtained from the supervised examples. The online regression oracle is implemented using PyTorch's Adam optimizer with log loss (recall that rewards are 0/1).

Hyperparameters. For each algorithm, we optimize its hyperparameters using random search (Bergstra & Bengio, 2012). Speccifically, hyperparameters are tuned by taking the best of 59 randomly selected configurations for a fixed seed (this seed is not used for evaluation). A seed determines both dataset shuffling, initialization of regressor parameters, and random choices made by any action sampling scheme.

Evaluation. We evaluate each algorithm on 32 seeds. All reported confidence intervals are 90% bootstrap CIs for the mean.

E.2. Practical Modification to Sampling Procedure in SpannerIGW

For experiments with SpannerIGW, we slightly modify the action sampling distribution so as to avoid computing the normalization constant λ. First, we modify the weighted embedding scheme given in Eq. (5) using the following expression:

$$\bar{\phi}(x_t, a) := \frac{\phi(x_t, a)}{\sqrt{1 + d + \frac{\gamma}{4d}\left(\widehat{f}_t(x_t, \widehat{a}_t) - \widehat{f}_t(x_t, a)\right)}}.$$

We obtain a $4d$-approximate optimal design for the reweighted embeddings by first computing a 2-approximate barycentric spanner $\mathcal{S}$, then taking $q_t^{\mathrm{opt}} := \mathrm{unif}(\mathcal{S})$. To proceed, let $\widehat{a}_t := \arg\max_{a \in \mathcal{A}} \widehat{f}(x_t, a)$ and $\bar{d} := |\mathcal{S} \cup \{\widehat{a}_t\}|$. We construct the sampling distribution $p_t \in \Delta(\mathcal{A})$ as follows:

- Set $p_t(a) := \frac{1}{\bar{d} + \frac{\gamma}{4d}\left(\widehat{f}_t(x_t, \widehat{a}_t) - \widehat{f}_t(x_t, a)\right)}$ for each $a \in \mathrm{supp}(\mathcal{S})$.
- Assign remaining probability mass to $\widehat{a}_t$.

With a small modification to the proof of Lemma 4, one can show that this construction certifies that $\mathsf{dec}_\gamma(\mathcal{F}) = O\left(\frac{d^2}{\gamma}\right)$. Thus, the regret bound in Theorem 2 holds up to a constant factor. Similarly, with a small modification to the proof of Theorem 3, we can also show that —with respect to this new embedding—Algorithm 3 has $O((\mathcal{T}_{\mathsf{Opt}} \cdot d^3 + d^4) \cdot \log^2\left(\frac{d+\gamma/d}{r}\right))$ runtime and $O(\mathcal{M}_{\mathsf{Opt}} + d^2 + d\log\left(\frac{d+\gamma/d}{r}\right))$ memory.

E.3. Timing Information

Table 4 contains timing information the oneshotwiki-14031 dataset with a bilinear model. The CPU timings are most relevant for practical scenarios such as information retrieval and recommendation systems, while the GPU timings are relevant for scenarios where simulation is possible. Timings for SpannerGreedy do not include the one-time cost to compute the spanner set. Timings for all algorithms use precomputed context and action embeddings. For all but algorithms but SpannerIGW, timings reflect the major bottleneck of computing the argmax action, since all subsequent steps take $O(1)$ time with respect to $|\mathcal{A}|$. In particular, SquareCB is implemented using rejection sampling, which does not require explicit construction of the action distribution. For SpannerIGW, the additional overhead is due to the time required to construct an approximate optimal design for each example.

Table 4: Per-example inference timings for oneshotwiki-14031. CPU timings use batch size 1 on an Azure STANDARD_D4_V2 machine. GPU timings use batch size 1024 on an Azure STANDARD_NC6S_V2 (Nvidia P100-based) machine.

Algorithm	CPU	GPU
ε-Greedy	2 ms	10 μs
SpannerGreedy	2 ms	10 μs
SquareCB	2 ms	10 μs
SpannerIGW	25 ms	180 μs

E.4. Additional Figures

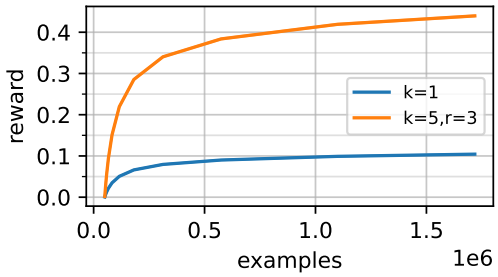

Figure 1: amazon-3m results for SpannerGreedy. Confidence intervals are rendered, but are but too small to visualize. For $(k = 1)$, the final CI is $[0.1041, 0.1046]$, and for $(k = 5, r = 3)$, the final CI is $[0.438, 0.440]$.