Breaking the Linear Iteration Cost Barrier for Some Well-known Conditional Gradient Methods Using MaxIP Data-structures

Anonymous Author(s)
Affiliation
Address
email

Abstract

Conditional gradient methods (CGM) are widely used in modern machine learning. CGM's overall running time usually consists of two parts: the number of iterations and the cost of each iteration. Most efforts focus on reducing the number of iterations as a means to reduce the overall running time. In this work, we focus on improving the per iteration cost of CGM. The bottleneck step in most CGM is maximum inner product search (MaxIP), which requires a linear scan over the parameters. In practice, approximate MaxIP data-structures are found to be helpful heuristics. However, theoretically, nothing is known about the combination of approximate MaxIP data-structures and CGM. In this work, we answer this question positively by providing a formal framework to combine the locality sensitive hashing type approximate MaxIP data-structures with CGM algorithms. As a result, we show the first sublinear time algorithm for many fundamental optimization algorithms, e.g., Frank-Wolfe, Herding algorithm, and policy gradient.

1 Introduction

Conditional gradient methods (CGM), such as Frank-Wolfe and its variants, are well-known optimization approaches that have been extensively used in modern machine learning. For example, CGM has been applied to kernel methods [1, 2], structural learning [3] and online learning [4, 5, 6].

Running Time Acceleration in Optimization: Recent years have witnessed the success of large-scale machine learning models on vast amounts of data. In this learning paradigm, the overhead of most successful models is dominated by the optimization process [7, 8]. Therefore, reducing the running time of the optimization algorithm is of practical importance. The total running time in optimization can be decomposed into two components: (1) the number of iterations towards convergence, (2) the cost spent in each iteration. Reducing the number of iterations requires a better understanding of the geometric proprieties of the problem at hand and how to create better potential functions to analyze the progress of the algorithm [9, 10, 11, 12, 13, 14, 15]. Reducing the cost spent per iteration usually condenses to design problem-specific discrete data-structures. In the last few years, we have seen a remarkable growth of using data-structures to reduce iteration cost [16, 17, 18, 19, 20, 21, 22, 23, 24].

MaxIP Data-structures for Iteration Cost Reduction: A well-known strategy in optimization, with CGM, is to perform a greedy search over the weight vectors [9, 10, 13, 16, 25] or training samples [26, 27] in each iteration. In this situation, the cost spent in each iteration is linear in the number of parameters. In practical machine learning, recent works [28, 29, 30, 31, 32] formulate this linear cost in iterative algorithms as an approximate maximum inner product search problem (MaxIP) and speed up the amortized cost per iteration via efficient data-structures from recent advances in approximate MaxIP [33, 34, 35, 36, 37, 38, 39, 40, 41, 42]. In approximate MaxIP data-structures, locality sensitive hashing (LSH) achieves promising performance with efficient random projection based preprocessing strategies [33, 34, 35, 36]. Thus, it is widely used in practice for cost reduction in optimization. [28] proposes an LSH based gradient sampling approach that reduces the total empirical running time of the adaptive gradient descent. [29] formulates the forward propagation of deep neural network as a MaxIP problem and uses LSH to select a subset of neurons for backpropagation. Therefore, the total running time of neural network training could be reduced to sublinear in the number of neurons. [31] extends this idea with system-level design for further acceleration, and [30] modifies the LSH with learning and achieves promising acceleration in attention-based language models. [32] formulates the greedy step in iterative machine teaching (IMT) as a MaxIP problem and scale IMT to large datasets with LSH.

Challenges of Sublinear Iteration Cost CGM: Despite the practical success of cost-efficient iterative algorithms with approximate MaxIP data-structure, the theoretical analysis of its combination with CGM is not well-understood. In this paper, we focus on this combination and target at answering the following questions: (1) how to transform the iteration step of CGM algorithms into an approximate MaxIP problem? (2) how does the approximate error in MaxIP affect CGM in the total number of iterations towards convergence? (3) how to adapt approximate MaxIP data structure for iterative CGM algorithms?

Our Contributions: We propose a theoretical formulation for combining approximate MaxIP and convergence guarantees of CGM. In particular, we start with the popular Frank-Wolfe algorithm over the convex hull where the direction search in each iteration is an approximate MaxIP problem. Next, we propose a sublinear iteration cost Frank-Wolfe algorithm using LSH type MaxIP data-structures. We then analyze the trade-off of approximate MaxIP and its effect on the number of iterations needed by CGM to converge. We show that the approximation obtained via LSH results in only a constant multiplicative factor increase in the number of iterations. As a result, we retain the sub-linearly of LSH, with respect to the number of parameters, and at the same time retain the same asymptotic convergence as CGMs.

We summarize our complete contributions as follows.

- We give the first theoretical CGM formulation that achieves provable sublinear time cost per iteration. We also extend this result into Frank-Wolfe algorithm, Herding algorithm, and policy gradient method.
- We propose a pair of efficient transformations that formulate the direction search in Frank-Wolfe algorithm as a projected approximate MaxIP problem.
- We present the theoretical results that the proposed sublinear Frank-Wolfe algorithm asymptotically preserves the same order in the number of iterations towards convergence. Furthermore, we analyze the trade-offs between saving iteration cost and the increasing number of iterations to accelerate total running time.
- We identify the problems of LSH type approximate MaxIP for cost reduction in other popular CGM methods and propose corresponding solutions.

The following sections are organized as below: Section 2 introduces the related works on data-structures and optimization, Section 3 introduces our algorithm associated with the main statements convergence, Section 4 provide the proof sketch of the main statements, Section 5 presents the societal impact and Section 6 conclude the paper.

2 Related work

2.1 Maximum Inner Product Search for Machine Learning

Maximum Inner Product Search (MaxIP) is a fundamental problem with applications in machine learning. Given a query $x \in \mathbb{R}^d$ and a dataset $Y \subset \mathbb{R}^d$ with n vectors, MaxIP targets at searching for $y \in Y$ that maximize the inner product $x^\top y$. The naive MaxIP solution takes $O(dn)$ by comparing x with each $y \in Y$. To accelerate this procedure, various algorithms are proposed to reduce the running time of MaxIP [33, 34, 36, 35, 37, 38, 43, 44, 39, 45, 40, 41, 42]. We could categorize the MaxIP approaches into two categories: reduction methods and non-reduction methods. The reduction methods use transformations to transform approximate MaxIP to approximate nearest neighbor search (ANN) and solve it with ANN data-structures. One of the popular data-structure is to use locality sensitive hashing [46, 47].

Definition 2.1 (Locality Sensitive Hashing). *Let $\overline{c}$ denote a parameter such that $\overline{c} > 1$. Let p_1, p_2 denote two parameters such that $0 < p_2 < p_1 < 1$. A family $\mathcal{H}$ is called $(r, \overline{c} \cdot r, p_1, p_2)$-sensitive if and only if, for any two point $x, y \in \mathbb{R}^d$, h chosen uniformly from $\mathcal{H}$ satisfies the following:*

- *if* $\|x-y\|_2 \leq r$, *then* $\Pr_{h\sim\mathcal{H}}[h(x)=h(y)] \geq p_1$,
- *if* $\|x-y\|_2 \geq \overline{c}\cdot r$, *then* $\Pr_{h\sim\mathcal{H}}[h(x)=h(y)] \leq p_2$.

Here we define the LSH functions for euclidean distance. LSH functions could be used for search in cosine [48, 49] or Jaccard similarity [50, 51]. [33] first show that MaxIP could be solved by ℓ_2 LSH and asymmetric transformations. After that, [34, 36, 35, 43] propose a series of methods to solve MaxIP via LSH functions for other distance measures. Besides LSH, graph-based ANN approaches [38] could also be used after reduction. On the other hand, the non-reduction method direct builds data-structures for approximate MaxIP. [37, 42] uses quantization to approximate the inner product distance and build codebooks for efficient approximate MaxIP. [38, 44] proposes a greedy algorithm for approximate MaxIP under computation budgets. [39, 40, 41] direct construct navigable graphs that achieve the state-of-the-art empirical performance.

Recently, there is a remarkable growth in applying data-structures for machine learning [52, 53, 54]. Following the paradigm, approximate MaxIP data-structures have been applied to overcome the efficiency bottleneck of various machine learning algorithms. [38] formulates the inference of neural network with a wide output layer as a MaxIP problem and uses a graph-based approach to reduce the inference time. In same task, [55] proposes a learnable LSH data-structures that further improves the inference efficiency with less energy consumption. In neural network training, [29, 30, 31] uses approximate MaxIP to retrieve interested neurons for backpropagation. In this way, the computation overhead of gradient update in neural networks could be reduced. In large-scale linear models, [28] uses approximate MaxIP data-structures to retrieve the samples with large gradient norm and perform standard gradient descent, which improves the total running time for stochastic gradient descent. [32] proposes a scalable machine teaching algorithm that enables iterative teaching in large-scale datasets. In bandit problem, [56]. proposes an LSH based algorithm that achieves linear bandits algorithms with sublinear time complexity.

Despite the promising empirical results, there is little theoretical analysis on approximate MaxIP for machine learning. We summarize the major reasons as: (1) Besides LSH, the other approximate MaxIP data-structures do not provide theoretical guarantees on time and space complexity. (2) Current approaches treat data-structures and learning dynamics separately. There is no joint analysis on the effect of approximate MaxIP for machine learning.

2.2 Projection-free Optimization

Frank-Wolfe algorithm [25] is a projection-free optimization method with wide applications in convex [9, 10] and non-convex optimizations [11, 12]. The procedure of Frank-Wolfe algorithm could be summarized two steps: (1) given the gradient, find the vector in the feasible domain that has the maximum inner product. (2) update the current weight with the retrieved vector. Formally, given a function $g:\mathbb{R}^d\to\mathbb{R}$ over a convex set S, starting from a initial weight w^0 the Frank-Wolfe algorithm performs update the weight with learning rate η following:

$$s^t \leftarrow \arg\min_{s\in S}\langle s, \nabla g(w^t)\rangle$$
$$w^{t+1} \leftarrow (1-\eta_t)\cdot w^t + \eta_t \cdot s^t.$$

Previous literature focuses on reducing the number of iterations for Frank-Wolfe algorithm over specific domains such as ℓ_p balls [9, 10, 13, 14]. There exists less work discussing the reduction of iteration cost in the iterative procedure of Frank-Wolfe algorithm. In this work, we focus on the Frank-Wolfe algorithm over the convex hull of a finite feasible set. This formulation is more general and it includes recent Frank-Wolfe applications in probabilistic modeling [1, 2], structural learning [3] and policy optimization [5].

3 Our Sublinear Iteration Cost Algorithm

In this section, we formally present our results on the sublinear iteration cost CGM algorithms. We start with the preliminary definitions of the objective function. Then, we present the results on the number of iterations and cost per iterations for our sublinear CGM algorithms to converge.

3.1 Preliminaries

We provide the notations and settings for this paper. We start with basic notations for this paper. For a positive integer n, we use $[n]$ to denote the set $\{1,2,\cdots,n\}$. For a vector x, we use $\|x\|_2 := (\sum_{i=1}^n x_i^2)^{1/2}$ to denote its ℓ_2 norm.

We say a function convex if

$$L(x) \geq L(y) + \langle \nabla L(y), x - y \rangle.$$

We say a function is β-smooth if

$$L(y) \leq L(x) + \langle \nabla L(x), y - x \rangle + \frac{\beta}{2} \|y - x\|_2^2.$$

Given a set $A = \{x_i\}_{i \in [n]} \subset \mathbb{R}^d$, we say its convex hull $\mathcal{B}(A)$ is the collection of all finite linear combinations y that satisfies $y = \sum_{i \in [n]} a_i \cdot x_i$, where $a_i \in (0, 1)$ for all $i \in [n]$ and $\sum_{i \in [n]} a_i = 1$. Let $D_{\max}$ denotes the maximum diameter of $\mathcal{B}(A)$ so that $\|x - y\|_2 \leq D_{\max}$ for all $(x, y) \in \mathcal{B}(A)$. We present the detailed definitions in Appendix A.

Next, we present the settings of our work. Let $S \subset \mathbb{R}^D$ denotes a n-point finite set. Given a convex and β-smooth function $g : \mathbb{R}^d \to \mathbb{R}$ defined over the convex hull $B(S)$. Our goal is to find a $w \in B(S)$ that minimizes $g(w)$. Given large n in the higher dimension, the dominant complexity of iteration cost is the finding the MaxIP of $\nabla g(w)$ with respect to S. In this setting, the fast learning rate of Frank-Wolfe in ℓ_p balls [9, 13, 16] could not be achieved. We present the detailed problem setting the Frank-Wolfe algorithm in Appendix C.

3.2 Our results

We present our main results with comparison to the original algorithm in Table 2. From the table, we show that with near-linear preprocessing time, our algorithms maintain the same number of iterations towards convergence while reducing the cost spent in each iteration to sublinear in the number of possible parameters.

	Statement	Preprocess	#Iters	Cost per iter
Frank-Wolf	[9]	0	$O(\beta D_{\max}^2/\epsilon)$	$O(dn + \mathcal{T}_g)$
Ours	Theorem 3.1	$dn^{1+o(1)}$	$O(\beta D_{\max}^2/\epsilon)$	$O(dn^\rho + \mathcal{T}_g)$
Herding	[1]	0	$O(D_{\max}^2/\epsilon)$	$O(dn)$
Ours	Theorem 3.2	$dn^{1+o(1)}$	$O(D_{\max}^2/\epsilon)$	$O(dn^\rho)$
Policy gradient	[5]	0	$O(\frac{\beta D_{\max}^2}{\epsilon^2(1-\gamma)^3\mu_{\min}^2})$	$O(dn + \mathcal{T}_Q)$
Ours	Theorem 3.3	$dn^{1+o(1)}$	$O(\frac{\beta D_{\max}^2}{\epsilon^2(1-\gamma)^3\mu_{\min}^2})$	$O(dn^\rho + \mathcal{T}_Q)$

Table 1: Comparison between classical algorithm and our sublinear time algorithm. We compare our algorithm with Frank-Wolfe in: (1) "Frank-Wolfe" denotes Frank-Wolfe algorithm [9] for convex functions over a convex hull. Let $\mathcal{T}_g$ denotes the time for evaluating the gradient for any parameter. (2) "Herding" denotes kernel herding algorithm [1] (3) "Policy gradient" denotes the projection free policy gradient method [5]. Let $\mathcal{T}_Q$ the time for evaluating the policy gradient for any parameter. Let $\gamma \in (0, 1)$ denotes the discount factor. Note that n is the number of possible parameters. $n^{o(1)}$ is smaller than n^c for any constant $c > 0$. Let $\rho \in (0, 1)$ denote a fixed parameter determined by LSH data-structure. The failure probability of our algorithm is $1/\operatorname{poly}(n)$. β is the smoothness factor. $D_{\max}$ denotes the maximum diameter of the coonvex hull.

Next, we introduce the statement of each sublinear iteration cost algorithm. We start by introducing our result for improving the running time of Frank-Wolfe.

Theorem 3.1 (Sublinear time Frank-Wolfe, informal of Theorem D.1)**.** *Let $g : \mathbb{R}^d \to \mathbb{R}$ denotes a convex and β-smooth function. Let the complexity of calculating $\nabla g(x)$ to be $\mathcal{T}_g$. Let $S \subset \mathbb{R}^d$ denotes a set of n points, and $\mathcal{B} \subset \mathbb{R}^d$ is the convex hull of S with maximum diameter $D_{\max}$. Let $\rho \in (0, 1)$ denote a fixed parameter. For any parameters ϵ, δ, there is an iterative algorithm (Algorithm 2) with that takes $O(dn^{1+o(1)})$ time in pre-processing, takes $T = O(\beta D_{\max}^2/\epsilon)$ iterations and $O(dn^\rho + \mathcal{T}_g)$ cost per iteration, starts from a random w^0 from $\mathcal{B}$ as initialization point,and outputs $w^T \in \mathbb{R}^d$ from $\mathcal{B}$ such that*

$$g(w^T) - \min_{w \in \mathcal{B}} g(w) \leq \epsilon,$$

holds with probability at least $1 - 1/\operatorname{poly}(n)$.

Next, we show our main result for the Herding algorithm. Herding algorithm is widely applied in kernel methods [57]. [1] shows that the Herding algorithm is equivalent to a conditional gradient method with the least-squares loss function. Therefore, we extend our results and obtain the following statement.

Theorem 3.2 (Sublinear time Herding algorithm, informal version of Theorem E.3)**.** *Let $\mathcal{X} \subset \mathbb{R}^d$ denote a feature set and $\Phi : \mathbb{R}^d \to \mathbb{R}^k$ denote a mapping. Let $D_{\max}$ denote the maximum diameter of $\Phi(\mathcal{X})$ and $\mathcal{B}(\Phi(\mathcal{X}))$ denote the convex hull of $\Phi(\mathcal{X})$. Given a distribution $p(x)$ over $\mathcal{X}$, we denote $\mu = \mathbb{E}_{x\sim p(x)}[\Phi(x)]$. Let $\rho \in (0, 1)$ denotes a fixed parameter. For any parameters ϵ, δ, there is an iterative algorithm (Algorithm 3) that takes $O(dn^{1+o(1)})$ time in pre-processing, takes $T = O(D_{\max}^2/\epsilon)$ iterations and $O(dn^\rho)$ cost per iteration, starts from a random w^0 from $\mathcal{B}$ as initialization point, and outputs $w^T \in \mathbb{R}^d$ from $\mathcal{B}(\Phi(\mathcal{X}))$ such that*

$$\frac{1}{2}\|w^T - \mu\|_2^2 - \min_{w\in\mathcal{B}} \frac{1}{2}\|w - \mu\|_2^2 \le \epsilon,$$

holds with probability at least $1 - 1/\operatorname{poly}(n)$.

Finally, we present our result for policy gradient. Policy gradient [58] is a popular algorithm with wide applications in robotics [59] and recommendation [60]. [5] proposes a provable Frank-Wolfe method that maximizes the reward functions with policy gradient. However, the optimization requires a linear scan over all possible actions, which is unscalable in complex environments. We propose an efficient Frank-Wolfe algorithm with per iteration cost sublinear in the number of actions. Our statement is presented as below.

Theorem 3.3 (Sublinear time policy gradient, informal version of Theorem F.3)**.** *Let $\mathcal{T}_Q$ denotes the time for computing the policy graident. Let $D_{\max}$ denotes the maximum diameter of action space and β is a constant. Let $\gamma \in (0, 1)$. Let $\rho \in (0, 1)$ denotes a fixed parameter. Let $\mu_{\min}$ denotes the minimal density of states in $\mathcal{S}$. There is an iterative algorithm (Algorithm 5) that spends $O(dn^{1+o(1)})$ time in preprocessing, takes $O(\frac{\beta D_{\max}^2}{\epsilon^2(1-\gamma)^3\mu_{\min}^2})$ iterations and $O(dn^\rho + \mathcal{T}_Q)$ cost per iterations, start from a random point π_θ^0 as initial point, and output π_θ^T that have the average gap $\sqrt{\sum_{s\in\mathcal{S}} g_T(s)^2} < \epsilon$ holds with probability at least $1-1/\operatorname{poly}(n)$, where $g_T(s)$ is defined in Eq. (6).*

4 Proof Overview

We present the overview of proofs in this section. We start with introducing the efficient MaxIP data-structures. Next, we show how to transform the direction search in conditional gradient approach as a MaxIP problem. Finally, we provide proof sketches for each main statement in Section 3. The detailed proof is presented in the supplement material.

4.1 Approximate MaxIP Data-structures

We present the LSH data-structures for approximate MaxIP in this section. The detailed description is presented in Appendix A. We use the reduction-based approximate MaxIP method with LSH data-structure to achieve sublinear iteration cost. Note that we choose this method due to its clear theoretical guarantee on the retrieval results. It is well-known that an LSH data-structures is used for approximate nearest neighbor problem. The following definition of approximate nearest neighbor search is very standard in literature [61, 46, 47, 62, 63, 64, 65, 66, 67, 68, 69].

Definition 4.1 (Approximate Nearest Neighbor (ANN))**.** *Let $\overline{c} > 1$ and $r \in (0, 2)$. Given an n-point dataset $P \subset \mathbb{S}^{d-1}$ on the sphere, the goal of the $(\overline{c}, r)$-Approximate Near Neighbor problem (*ANN*) is to build a data structure that, given a query $q \in \mathbb{S}^{d-1}$ with the promise that there exists a datapoint $p \in P$ with $\|p - q\|_2 \le r$ reports a datapoint $p' \in P$ within distance $\overline{c} \cdot r$ from q.*

In the iterative-type optimization algorithm, the cost per iteration could be dominated by the Approximate MaxIP problem (Definition 4.2), which is the dual problem of the $(\overline{c}, r)$-ANN.

Definition 4.2 (Approximate MaxIP)**.** *Let $c \in (0, 1)$ and $\tau \in (0, 1)$. Given an n-point dataset $Y \subset \mathbb{S}^{d-1}$, the goal of the (c, τ)-*MaxIP *is to build a data structure that, given a query $x \in \mathbb{S}^{d-1}$ with the promise that there exists a vector $y \in Y$ with $\langle x, y\rangle \ge \tau$, it reports a vector $z \in Y$ with similarity $\langle x, z\rangle \ge c \cdot \mathsf{MaxIP}(x, Y)$.*

Next, we present the the primal-dual connection between ANN and approximate MaxIP. Given to unit vectors $x, y \in \mathbb{R}^d$ with both norm equal to 1, $\|x - y\|_2^2 = 2 - 2\langle x, y\rangle$. Therefore, we could maximizing $\langle x, y\rangle$ by minimizing $\|x - y\|_2^2$. Based on this connection, we present how to solve (c, τ)-MaxIP using $(\overline{c}, r)$-ANN. We start with showing how to solve $(\overline{c}, r)$-ANN with LSH.

Theorem 4.3 ([66])**.** *Let $\overline{c} > 1$ and $r \in (0, 2)$. The $(\overline{c}, r)$-ANN on a unit sphere $\mathcal{S}^{d-1}$ can be solved in query time $O(d \cdot n^\rho)$, where $\rho \in (0, 1)$, using* LSH *with both preprocessing time and space in $O(n^{1+o(1)} + dn)$.*

Next, we solve (c, τ)-MaxIP by solving $(\overline{c}, r)$-ANN using Theorem 4.3. We have

Corollary 4.4 (An informal statement of Corollary B.1)**.** *Let $c \in (0, 1)$ and $\tau \in (0, 1)$. The (c, τ)-*MaxIP *on a unit sphere $\mathcal{S}^{d-1}$ can be solved in query time $O(d \cdot n^\rho)$, where $\rho \in (0, 1)$, using* LSH *with both preprocessing time and space in $O(dn^{1+o(1)})$*

In our work, we consider a generalized form of approximate MaxIP, denoted as projected approximate MaxIP.

Definition 4.5 (Projected approximate MaxIP)**.** *Let $\phi, \psi : \mathbb{R}^d \to \mathbb{R}^k$ denotes two transforms. Given an n-point dataset $Y \subset \mathbb{R}^d$ so that $\psi(Y) \subset \mathbb{S}^{d-1}$, the goal of the (c, ϕ, ψ, τ)-*MaxIP *is to build a data structure that, given a query $x \in \mathbb{R}^d$ and $\phi(x) \in \mathbb{S}^{k-1}$ with the promise that $\max_{y \in Y}\langle \phi(x), \psi(y)\rangle \geq \tau$, it reports a datapoint $z \in Y$ with similarity $\langle \phi(x), \psi(z)\rangle \geq c \cdot \mathsf{MaxIP}(\phi(x), \psi(Y))$.*

For details of space-time trade-offs, please refer to Appendix B. In the following sections, we would show how to use projected approximate MaxIP to accelerate the optimization algorithm by reducing the cost per iteration.

4.2 Efficient Transformations

We have learned from Section 4.1 that (c, τ)-MaxIP on a unit sphere $\mathcal{S}^{d-1}$ using LSH for ANN. Therefore, the next step is to transform the direction search procedure in iterative optimization algorithm into a MaxIP on a unit sphere. To achieve this, we formulate the direction search as a projected approximate MaxIP (see Definition A.5). We start with presenting a pair of transformation $\phi_0, \psi_0 : \mathbb{R}^d \to \mathbb{R}^{d+1}$ such that, given a function $g : \mathbb{R}^d \to \mathbb{R}$, for any x, y in a convex set $\mathcal{K}$, we have

$$\phi_0(x) := [\nabla g(x)^\top, x^\top \nabla g(x)]^\top, \quad \psi_0(y) := [-y^\top, 1]^\top. \tag{1}$$

In this way, we show that

$$\begin{aligned} \langle y - x, \nabla g(x)\rangle &= -\langle \phi_0(x), \psi_0(y)\rangle, \\ \arg\min_{y \in Y}\langle y - x, \nabla g(x)\rangle &= \arg\max_{y \in Y}\langle \phi_0(x), \psi_0(y)\rangle \end{aligned} \tag{2}$$

Therefore, we could transform the direction search problem into a MaxIP problem.

Next, we present a standard transformations [36] that connects the MaxIP to ANN in unit sphere. For any $x, y \in \mathbb{R}^d$, we propose transformation $\phi_1, \psi_1 : \mathbb{R}^d \to \mathbb{R}^{d+2}$ such that

$$\begin{aligned} \phi_1(x) &= \begin{bmatrix} (D_x^{-1}x)^\top & 0 & \sqrt{1 - \|xD_x^{-1}\|_2^2} \end{bmatrix}^\top \\ \psi_1(y) &= \begin{bmatrix} (D_y^{-1}y)^\top & \sqrt{1 - \|yD_y^{-1}\|_2^2} & 0 \end{bmatrix}^\top \end{aligned} \tag{3}$$

Here D_x, D_y are the maximum diameter of x and y. Under these transformations, both $\phi_1(x)$ and $\psi_1(y)$ have norm 1 and $\arg\max_{y \in Y}\langle \phi_1(x), \psi_1(y)\rangle = \arg\max_{y \in Y}\langle x, y\rangle$.

Combining transformations in Eq. (1) and Eq. (3), we obtain query transform $\phi : \mathbb{R}^d \to \mathbb{R}^{d+3}$ with form $\phi(x) = \phi_1(\phi_0(x))$ and data transform $\phi : \mathbb{R}^d \to \mathbb{R}^{d+3}$ with form $\psi(y) = \psi_1(\psi_0(y))$. Using ϕ and ψ, we transform the direction search problem in optimization into a MaxIP in unit sphere.

Moreover, given a set $Y \subset \mathbb{R}^d$ and a query $x \in \mathbb{R}^d$, the solution z of (c, ϕ, ψ, τ)-MaxIP over (x, Y) has the propriety that $\langle z - x, \nabla g(x) \rangle \leq c \cdot \min_{y \in Y} \langle y - x, \nabla g(x) \rangle$. Thus, we could approximate the direction search with LSH based MaxIP data-structure.

Note that only MaxIP problem with positive inner product values could be solved by LSH. We found the direction search problem naturally satisfies this condition. We show that if g is convex, given a set $S \subset \mathbb{R}^d$, we have $\min_{s \in S} \langle \nabla g(x), s - x \rangle \leq 0$ for any $x \in \mathcal{B}(S)$, where $\mathcal{B}$ is the convex hull of S. Thus, $\max_{y \in Y} \langle \phi_0(x), \psi_0(y) \rangle$ is non-negative following Eq. (2).

4.3 Proof of Theorem 3.1

We present the proof sketch for Theorem 3.1 in this section. We refer the readers to Appendix D for the detailed proofs.

Let $g : \mathbb{R}^d \to \mathbb{R}$ denotes a convex and β-smooth function. Let the complexity of calculating $\nabla g(x)$ to be $\mathcal{T}_g$. Let $S \subset \mathbb{R}^d$ denotes a set of n points, and $\mathcal{B} \subset \mathbb{R}^d$ is the convex hull of S with maximum diameter $D_{\max}$. Let $\phi, \psi : \mathbb{R}^d \to \mathbb{R}^{d+3}$ denotes the tranformations defined in Section 4.2. Starting from a random vector $w^0 \in \mathcal{B}(S)$. Our sublinear Frank-Wolfe algorithm follows the update following rule that each step

$$\begin{aligned} s^t &\leftarrow (c, \phi, \psi, \tau)\text{-MaxIP of } w^t \text{ with respect to } S \\ w^{t+1} &\leftarrow w^t + \eta \cdot (s^t - w^t) \end{aligned}$$

We start with the upper bounding $\langle s^t - w^t, \nabla g(w^t) \rangle$. Because s^t is the (c, ϕ, ψ, τ)-MaxIP of w^t with respect to S, we have

$$\langle s^t - w^t, \nabla g(w^t) \rangle \leq c \min_{s \in S} \langle s - w^t, \nabla g(w^t) \leq c \langle w^* - w^t, \nabla g(w^t) \rangle \tag{4}$$

For convenient of the proof, for each t, we define $h_t = g(w^t) - g(w^*)$. Next, we upper bound h_{t+1} as

$$\begin{aligned} h_{t+1} &\leq g(w^t) + \eta_t \langle s^t - w^t, \nabla g(w^t) \rangle + \frac{\beta}{2} \eta_t^2 \| s^t - w^t \|_2^2 - g(w^*) \\ &\leq g(w^t) + c\eta_t \langle w^* - w^t, \nabla g(w^t) \rangle + \frac{\beta}{2} \eta_t^2 \| s^t - w^t \|_2^2 - g(w^*) \\ &\leq g(w^t) + c\eta_t \langle w^* - w^t, \nabla g(w^t) \rangle + \frac{\beta D_{\max}^2}{2} \eta_t^2 - g(w^*) \\ &\leq (1 - \eta_t) g(w^t) + c\eta_t g(w^*) + \frac{\beta D_{\max}^2}{2} \eta_t^2 - g(w^*) \\ &= (1 - c\eta_t) h_t + \frac{\beta D_{\max}^2}{2} \eta_t^2 \end{aligned} \tag{5}$$

where the first step follows from the definition of β-smoothness, the second step follows from Eq. (4), the third step follows from the definition of $D_{\max}$, the forth step follows from the convexity of g.

Let $\eta = \frac{2}{c(t+2)}$ and $A_t = \frac{t(t+1)}{2}$. Combining them with Eq.(5), we show that

$$\begin{aligned} A_{t+1} h_{t+1} - A_t h_t &= c^{-2} \frac{t+1}{t+2} \beta D_{\max}^2 \\ &< c^{-2} \beta D_{\max}^2 \end{aligned}$$

Using induction from 1 to t, we show that

$$A_t h_t < c^{-2} t \beta D_{\max}^2$$

Taken $A_t = \frac{t(t+1)}{2}$ into consideration, we have

$$h_t < \frac{2\beta D_{\max}^2}{c^2 (t+1)}$$

Given constant approximation ratio c, t should be in $O(\frac{\beta D_{\max}^2}{\epsilon})$ so that $h_t \leq \epsilon$.

Thus, we complete the proof.

Cost Per Iteration After we take $O(dn^{1+o(1)})$ preprocessing time, the cost per iteration consists three pairs: (1) it takes $\mathcal{T}_g$ to compute $\nabla g(w^t)$, (2) it takes $O(d)$ to perform transform ϕ and ψ, (3) it takes $O(dn^\rho)$ to retrieve s^t from LSH. Thus, the final cost per iteration would be $O(dn^\rho + \mathcal{T}_g)$.

Next, we show how to extend the proof to Herding problem. Following [1], we start with defining function $g = \frac{1}{2}\|w^T - \mu\|_2^2$. We show that this function g is a convex and 1-smooth function. Therefore, the herding algorithm is equivalent to the Frank-Wolfe Algorithm over function g. Using the proof of Theorem 3.1 with $\beta = 1$, we show that it takes $T = O(D_{\max}^2/\epsilon)$ iterations and $O(dn^\rho)$ cost per iteration to reach the ϵ optimal solution. Similar to Theorem 3.1, we show that the cost per iteration would be $O(dn^\rho)$ as it takes $O(d)$ to compute $\nabla g(w^t)$.

4.4 Proof of Theorem 3.3

We present the proof sketch for Theorem 3.3 in this section. We refer the readers to Appendix F for the detailed proofs.

In this paper, we focus on the action-constrained Markov Decision Process (ACMDP). In this setting, we are provided with a state $\mathcal{S} \in \mathbb{R}^k$ and action space $\mathcal{A} \in \mathbb{R}^d$. However, at each step $t \in \mathbb{N}$, we could only access a finite n-vector set of actions $\mathcal{C}(s) \subset \mathcal{A}$. Let us denote $D_{\max}$ as the maximum diameter of $\mathcal{A}$.

When you play with this ACMDP, the policy you choose is defined as $\pi_\theta(s) : \mathcal{S} \to \mathcal{A}$ with parameter θ. Meanwhile, there exists a reward function $r : \mathcal{S} \times \mathcal{A} \in [0, 1]$. Then, we define the Q function as below,

$$Q(s, a|\pi_\theta) = \mathbb{E}\Big[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t)|s_0 = s, a_0 = a, \pi_\theta\Big].$$

where $\gamma \in (0, 1)$ is a discount factor.

Given a state distribution μ, the objective of policy gradient is to maximize $J(\mu, \pi_\theta) = \mathbb{E}_{s\sim\mu, a\sim\pi_\theta}[Q(s, a|\pi_\theta)]$ via policy gradient [58] denoted as:

$$\nabla_\theta J(\mu, \pi_\theta) = \mathop{\mathbb{E}}_{s\sim d_\mu^\pi}\Big[\nabla_\theta \pi_\theta(s) \nabla_a Q(s, \pi_\theta(s)|\pi_\theta)|\Big].$$

[5] propose an iterative algorithm that perform MaxIP at each iteration k over actions to find

$$g_k(s) = \max_{a\in\mathcal{C}(s)} \langle a_s^k - \pi_\theta^k(s), \nabla_a Q(s, \pi_\theta^k(s)|\pi_\theta^k))\rangle. \tag{6}$$

In this work, we approximate Eq. (6) using (c, ϕ, ψ, τ)-MaxIP. Here define $\phi : \mathcal{S} \times \mathbb{R}^d \to \mathbb{R}^{d+1}$ and $\psi : \mathbb{R}^d \to \mathbb{R}^{d+1}$ as follows:

$$\phi(s, \pi_\theta^k) := [\nabla_a Q(s, \pi_\theta^k(s)|\pi_\theta^k)^\top, (\pi_\theta^k)^\top Q(s, \pi_\theta^k(s)|\pi_\theta^k)]^\top, \psi(a) := [a^\top, -1]^\top.$$

Then, for all $x, y \in \mathbb{R}^d$ we have $g_k(s) = \langle \phi(s, \pi_\theta^k), \psi(a)\rangle$. Note that we still require transformations in Eq. (3) to generate unit vectors.

Next, we show that if we retrieve an action $\widehat{a_s^k}$ using (c, ϕ, ψ, τ)-MaxIP, the gap $\widehat{g}_k(s)$ would be lower bounded by

$$\begin{aligned}\widehat{g}_k(s) &= \langle \widehat{a_s^k} - \pi_\theta^k(s), \nabla_a Q(s, \pi_\theta^k(s)|\pi_\theta^k))\rangle \\ &\geq c g_k(s)\end{aligned} \tag{7}$$

Combining Eq. (7) the standard induction in [5], we upper bound $\sum_{s\in\mathcal{S}} g_T(s)^2$ as

$$\sum_{s\in\mathcal{S}} g_T(s)^2 \leq \frac{1}{T+1} \frac{2\beta D_{\max}^2}{c^2(1-\gamma)^3 \mu_{\min}^2}. \tag{8}$$

where $\mu_{\min}$ denotes the minimal density of sates in $\mathcal{S}$ and β is the smoothness factor.

In this way, given a constant factor c, if we would like to minimize the gap $\sum_{s\in\mathcal{S}} g_T(s)^2 < \epsilon^2$, T should be $O(\frac{\beta D_{\max}^2}{\epsilon^2(1-\gamma)^3\mu_{\min}^2})$.

Cost Per Iteration After we take $O(dn^{1+o(1)})$ preprocessing time, the cost per iteration consists three pairs: (1) it takes $\mathcal{T}_Q$ to compute policy gradient, (2) it takes $O(d)$ to perform transform ϕ and ψ, (3) it takes $O(dn^\rho)$ to retrieve actions from LSH. Thus, the final cost per iteration would be $O(dn^\rho + \mathcal{T}_Q)$.

4.5 Quantization for Adaptive Queries

In optimization, the gradient computed in every iteration is not independent of each other. This would generate a problem for MaxIP data-structures. If we use a vector containing the gradients as query for MaxIP data-structures, the query failure probability in each iteration is not independent. Therefore, the total failure probability could not be union bounded. As previous MaxIP data-structures focus on the assumptions that queries are independent, the original failure analysis could not be directly applied.

In this work, we use a standard query quantization method to handle the adaptive query sequence in optimization. Given the known query space, we quantize it by lattices [70]. This quantization is close to the Voronoi diagrams. In this way, each query is located into a cell with a center vector. Next, we perform query using the center vector in the cell. Therefore, the failure probability of the MaxIP query sequence is equivalent to the probability that any center vector in the cell fails to retrieve its approximate MaxIP solution. As the centers of cells are independent, we could union bound the probability. On the other hand, as the maximum diameter of the cell is λ. This query quantization would introduce a λ additive error in the inner product retrieved. We refer the readers to Appendix G for the detailed quantization approach.

4.6 Optimizing Accuracy-Efficiency Trade-off

In this work, we show that by LSH based MaxIP data-structure, the cost for direction search is $O(dn^\rho)$, where $\rho \in (0,1)$. In Section D.2 of the supplementary material, we show that ρ is a function of constant c and parameter τ in approximate MaxIP (see Definition 4.2). Moreover, we also show in Section D.2 that LSH results in only a constant multiplicative factor increase in the number of iterations. Considering the cost per iteration and the number of iterations, we show that when our algorithms stop at the ϵ-optimal solution, LSH could achieve acceleration in the overall running time. Therefore, we could set c and τ parameter to balance the accuracy-efficiency trade-off of CGM to achieve the desired running time.

5 Potential Negative Societal Impact

This paper discusses the theoretical foundation of data-structures for conditional gradient methods. We believe that this paper does not have negative societal impact in the environment, privacy, and other domains.

6 Concluding Remarks

In this work, we present the first Frank-Wolfe algorithms that achieve sublinear linear time cost per iteration. We also extend this result into herding algorithm and policy gradient methods. We formulate the direction search in Frank-Wolfe algorithm as a projected approximate maximum inner product search problem with a pair of efficient transformations. Then, we use locality sensitive hashing data-structure to reduce the iteration cost into sublinear over number of possible parameters. Our theoretical analysis shows that the sublinear iteration cost Frank-Wolfe algorithm preserves the same order in the number of iterations towards convergence. Moreover, we analyze and optimize the trade-offs between saving iteration cost and increasing the number of iterations to achieve sublinear total running time. Furthermore, we identify the problems of existing MaxIP data-structures for cost reduction in iterative optimization algorithms and propose the corresponding solutions. We hope this work can be the starting point of future study on sublinear iteration cost algorithm for optimization.

References

[1] Francis Bach, Simon Lacoste-Julien, and Guillaume Obozinski. On the equivalence between herding and conditional gradient algorithms. *arXiv preprint arXiv:1203.4523*, 2012.

[2] Paxton Turner, Jingbo Liu, and Philippe Rigollet. A statistical perspective on coreset density estimation. In *International Conference on Artificial Intelligence and Statistics (AISTATS)*, pages 2512–2520, 2021.

[3] Simon Lacoste-Julien, Martin Jaggi, Mark Schmidt, and Patrick Pletscher. Block-coordinate frank-wolfe optimization for structural svms. In *International Conference on Machine Learning (ICML)*, pages 53–61, 2013.

[4] Robert M Freund, Paul Grigas, and Rahul Mazumder. An extended frank–wolfe method with "in-face" directions, and its application to low-rank matrix completion. *SIAM Journal on optimization*, 27(1):319–346, 2017.

[5] Jyun-Li Lin, Wei Hung, Shang-Hsuan Yang, Ping-Chun Hsieh, and Xi Liu. Escaping from zero gradient: Revisiting action-constrained reinforcement learning via frank-wolfe policy optimization. *arXiv preprint arXiv:2102.11055*, 2021.

[6] Elad Hazan and Satyen Kale. Projection-free online learning. In *29th International Conference on Machine Learning, ICML 2012*, pages 521–528, 2012.

[7] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[8] Tom B Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *arXiv preprint arXiv:2005.14165*, 2020.

[9] Martin Jaggi. Revisiting frank-wolfe: Projection-free sparse convex optimization. In *International Conference on Machine Learning (ICML)*, pages 427–435, 2013.

[10] Dan Garber and Elad Hazan. Faster rates for the frank-wolfe method over strongly-convex sets. In *International Conference on Machine Learning (ICML)*, pages 541–549, 2015.

[11] Sashank J Reddi, Suvrit Sra, Barnabás Póczos, and Alex Smola. Stochastic frank-wolfe methods for nonconvex optimization. In *2016 54th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*, pages 1244–1251. IEEE, 2016.

[12] Sashank J Reddi, Ahmed Hefny, Suvrit Sra, Barnabas Poczos, and Alex Smola. Stochastic variance reduction for nonconvex optimization. In *International conference on machine learning (ICML)*, pages 314–323, 2016.

[13] Zeyuan Allen-Zhu, Elad Hazan, Wei Hu, and Yuanzhi Li. Linear convergence of a frank-wolfe type algorithm over trace-norm balls. In *NIPS*, 2017.

[14] Qi Lei, Jiacheng Zhuo, Constantine Caramanis, Inderjit S Dhillon, and Alexandros G Dimakis. Primal-dual block generalized frank-wolfe. *Advances in Neural Information Processing Systems (NeurIPS)*, 32:13866–13875, 2019.

[15] Ruosong Wang, Peilin Zhong, Simon S Du, Russ R Salakhutdinov, and Lin F Yang. Planning with general objective functions: Going beyond total rewards. In *Annual Conference on Neural Information Processing Systems*, 2020.

[16] Yin Tat Lee, Zhao Song, and Qiuyi Zhang. Solving empirical risk minimization in the current matrix multiplication time. In *COLT*. https://arxiv.org/pdf/1905.04447, 2019.

[17] Shunhua Jiang, Zhao Song, Omri Weinstein, and Hengjie Zhang. Faster dynamic matrix inverse for faster lps. In *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing (STOC)*. arXiv preprint arXiv:2004.07470, 2021.

[18] Zhao Song and Zheng Yu. Oblivious sketching-based central path method for solving linear programming problems. In *38th International Conference on Machine Learning (ICML)*, 2021.

[19] Jan van den Brand, Binghui Peng, Zhao Song, and Omri Weinstein. Training (overparametrized) neural networks in near-linear time. In *12th Innovations in Theoretical Computer Science Conference (ITCS)*, 2021.

[20] Michael B Cohen, Yin Tat Lee, and Zhao Song. Solving linear programs in the current matrix multiplication time. In *STOC*, 2019.

[21] Jan van den Brand, Yin Tat Lee, Aaron Sidford, and Zhao Song. Solving tall dense linear programs in nearly linear time. In *Proceedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing (STOC)*, pages 775–788, 2020.

[22] Sally Dong, Yin Tat Lee, and Guanghao Ye. A nearly-linear time algorithm for linear programs with small treewidth: A multiscale representation of robust central path. In *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing (STOC)*. arXiv preprint arXiv:2011.05365, 2021.

[23] Jan van den Brand. A deterministic linear program solver in current matrix multiplication time. In *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 259–278. SIAM, 2020.

[24] Jan van den Brand, Yin-Tat Lee, Danupon Nanongkai, Richard Peng, Thatchaphol Saranurak, Aaron Sidford, Zhao Song, and Di Wang. Bipartite matching in nearly-linear time on moderately dense graphs. In *2020 IEEE 61st Annual Symposium on Foundations of Computer Science (FOCS)*, pages 919–930. IEEE, 2020.

[25] Marguerite Frank, Philip Wolfe, et al. An algorithm for quadratic programming. *Naval research logistics quarterly*, 3(1-2):95–110, 1956.

[26] Weiyang Liu, Bo Dai, Ahmad Humayun, Charlene Tay, Chen Yu, Linda B Smith, James M Rehg, and Le Song. Iterative machine teaching. In *International Conference on Machine Learning*, pages 2149–2158. PMLR, 2017.

[27] Weiyang Liu, Bo Dai, Xingguo Li, Zhen Liu, James Rehg, and Le Song. Towards black-box iterative machine teaching. In *International Conference on Machine Learning*, pages 3141–3149. PMLR, 2018.

[28] Beidi Chen, Yingchen Xu, and Anshumali Shrivastava. Fast and accurate stochastic gradient estimation. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems (NeurIPS)*, 2019.

[29] Beidi Chen, Tharun Medini, James Farwell, sameh gobriel, Charlie Tai, and Anshumali Shrivastava. Slide : In defense of smart algorithms over hardware acceleration for large-scale deep learning systems. In I. Dhillon, D. Papailiopoulos, and V. Sze, editors, *Proceedings of Machine Learning and Systems*, pages 291–306, 2020.

[30] Beidi Chen, Zichang Liu, Binghui Peng, Zhaozhuo Xu, Jonathan Lingjie Li, Tri Dao, Zhao Song, Anshumali Shrivastava, and Christopher Re. MONGOOSE: A learnable LSH framework for efficient neural network training. In *International Conference on Learning Representations (ICLR)*, 2021.

[31] Shabnam Daghaghi, Nicholas Meisburger, Mengnan Zhao, and Anshumali Shrivastava. Accelerating slide deep learning on modern cpus: Vectorization, quantizations, memory optimizations, and more. *Proceedings of Machine Learning and Systems*, 3, 2021.

[32] Zhaozhuo Xu, Beidi Chen, Chaojian Li, Weiyang Liu, Le Song, Yingyan Lin, and Anshumali Shrivastava. Locality sensitive teaching. Technical report, Rice University, 2021.

[33] Anshumali Shrivastava and Ping Li. Asymmetric lsh (alsh) for sublinear time maximum inner product search (mips). *Advances in Neural Information Processing Systems (NIPS)*, pages 2321–2329, 2014.

[34] Anshumali Shrivastava and Ping Li. Improved asymmetric locality sensitive hashing (alsh) for maximum inner product search (mips). In *Proceedings of the Thirty-First Conference on Uncertainty in Artificial Intelligence (UAI)*, pages 812–821, 2015.

[35] Anshumali Shrivastava and Ping Li. Asymmetric minwise hashing for indexing binary inner products and set containment. In *Proceedings of the 24th international conference on world wide web (WWW)*, pages 981–991, 2015.

[36] Behnam Neyshabur and Nathan Srebro. On symmetric and asymmetric lshs for inner product search. In *International Conference on Machine Learning (ICML)*, pages 1926–1934. PMLR, 2015.

[37] Ruiqi Guo, Sanjiv Kumar, Krzysztof Choromanski, and David Simcha. Quantization based fast inner product search. In *Artificial Intelligence and Statistics (AISTATS)*, pages 482–490. PMLR, 2016.

[38] Hsiang-Fu Yu, Cho-Jui Hsieh, Qi Lei, and Inderjit S Dhillon. A greedy approach for budgeted maximum inner product search. In *Proceedings of the 31st International Conference on Neural Information Processing Systems (NIPS)*, pages 5459–5468, 2017.

[39] Stanislav Morozov and Artem Babenko. Non-metric similarity graphs for maximum inner product search. *Advances in Neural Information Processing Systems (NeurIPS)*, 31:4721–4730, 2018.

[40] Zhixin Zhou, Shulong Tan, Zhaozhuo Xu, and Ping Li. Möbius transformation for fast inner product search on graph. *Advances in Neural Information Processing Systems (NeurIPS)*, 32, 2019.

[41] Shulong Tan, Zhixin Zhou, Zhaozhuo Xu, and Ping Li. On efficient retrieval of top similarity vectors. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 5239–5249, 2019.

[42] Ruiqi Guo, Philip Sun, Erik Lindgren, Quan Geng, David Simcha, Felix Chern, and Sanjiv Kumar. Accelerating large-scale inference with anisotropic vector quantization. In *International Conference on Machine Learning (ICML)*, pages 3887–3896. PMLR, 2020.

[43] Xiao Yan, Jinfeng Li, Xinyan Dai, Hongzhi Chen, and James Cheng. Norm-ranging lsh for maximum inner product search. *Advances in Neural Information Processing Systems (NeurIPS)*, 31:2952–2961, 2018.

[44] Qin Ding, Hsiang-Fu Yu, and Cho-Jui Hsieh. A fast sampling algorithm for maximum inner product search. In *The 22nd International Conference on Artificial Intelligence and Statistics (AISTATS)*, pages 3004–3012. PMLR, 2019.

[45] Minjia Zhang, Xiaodong Liu, Wenhan Wang, Jianfeng Gao, and Yuxiong He. Navigating with graph representations for fast and scalable decoding of neural language models. *arXiv preprint arXiv:1806.04189*, 2018.

[46] Piotr Indyk and Rajeev Motwani. Approximate nearest neighbors: towards removing the curse of dimensionality. In *Proceedings of the thirtieth annual ACM symposium on Theory of computing (STOC)*, pages 604–613, 1998.

[47] Mayur Datar, Nicole Immorlica, Piotr Indyk, and Vahab S Mirrokni. Locality-sensitive hashing scheme based on p-stable distributions. In *Proceedings of the twentieth annual symposium on Computational geometry (SoCG)*, pages 253–262, 2004.

[48] Moses S Charikar. Similarity estimation techniques from rounding algorithms. In *Proceedings of the thiry-fourth annual ACM symposium on Theory of computing*, pages 380–388, 2002.

[49] Xiaoyun Li and Ping Li. Random projections with asymmetric quantization. *Advances in Neural Information Processing Systems*, 32:10858–10867, 2019.

[50] Ping Li, Xiaoyun Li, and Cun Hui Zhang. Re-randomized densification for one permutation hashing and bin-wise consistent weighted sampling. *Advances in Neural Information Processing Systems*, 32, 2019.

[51] Xiaoyun Li and Ping Li. Rejection sampling for weighted jaccard similarity revisited. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, 2021.

[52] Arturs Backurs, Piotr Indyk, and Tal Wagner. Space and time efficient kernel density estimation in high dimensions. *NeurIPS*, 2019.

[53] Amir Zandieh, Navid Nouri, Ameya Velingker, Michael Kapralov, and Ilya Razenshteyn. Scaling up kernel ridge regression via locality sensitive hashing. In *International Conference on Artificial Intelligence and Statistics*, pages 4088–4097. PMLR, 2020.

[54] Arturs Backurs, Yihe Dong, Piotr Indyk, Ilya Razenshteyn, and Tal Wagner. Scalable nearest neighbor search for optimal transport. In *International Conference on Machine Learning*, pages 497–506. PMLR, 2020.

[55] Zichang Liu, Zhaozhuo Xu, Alan Ji, Jonathan Li, Beidi Chen, and Anshumali Shrivastava. Climbing the wol: Training for cheaper inference. *arXiv preprint arXiv:2007.01230*, 2020.

[56] Shuo Yang, Tongzheng Ren, Sanjay Shakkottai, Eric Price, Inderjit S Dhillon, and Sujay Sanghavi. Linear bandit algorithms with sublinear time complexity. *arXiv preprint arXiv:2103.02729*, 2021.

[57] Yutian Chen, Max Welling, and Alex Smola. Super-samples from kernel herding. *arXiv preprint arXiv:1203.3472*, 2012.

[58] David Silver, Guy Lever, Nicolas Heess, Thomas Degris, Daan Wierstra, and Martin Riedmiller. Deterministic policy gradient algorithms. In *International conference on machine learning (ICML)*, pages 387–395, 2014.

[59] Jan Peters and Stefan Schaal. Policy gradient methods for robotics. In *2006 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2219–2225. IEEE, 2006.

[60] Haokun Chen, Xinyi Dai, Han Cai, Weinan Zhang, Xuejian Wang, Ruiming Tang, Yuzhou Zhang, and Yong Yu. Large-scale interactive recommendation with tree-structured policy gradient. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 3312–3320, 2019.

[61] Sunil Arya and David M Mount. Approximate nearest neighbor queries in fixed dimensions. In *SODA*, volume 93, pages 271–280. Citeseer, 1993.

[62] Alexandr Andoni, Piotr Indyk, Huy L Nguyen, and Ilya Razenshteyn. Beyond locality-sensitive hashing. In *Proceedings of the twenty-fifth annual ACM-SIAM symposium on Discrete algorithms*, pages 1018–1028. SIAM, 2014.

[63] Alexandr Andoni, Piotr Indyk, TMM Laarhoven, Ilya Razenshteyn, and Ludwig Schmidt. Practical and optimal lsh for angular distance. In *Advances in Neural Information Processing Systems (NIPS)*, pages 1225–1233. Curran Associates, 2015.

[64] Alexandr Andoni and Ilya Razenshteyn. Optimal data-dependent hashing for approximate near neighbors. In *Proceedings of the forty-seventh annual ACM symposium on Theory of computing (STOC)*, pages 793–801, 2015.

[65] Piotr Indyk and Tal Wagner. Approximate nearest neighbors in limited space. In *Conference On Learning Theory*, pages 2012–2036. PMLR, 2018.

[66] Alexandr Andoni, Thijs Laarhoven, Ilya Razenshteyn, and Erik Waingarten. Optimal hashing-based time-space trade-offs for approximate near neighbors. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 47–66. SIAM, 2017.

[67] Alexandr Andoni, Piotr Indyk, and Ilya Razenshteyn. Approximate nearest neighbor search in high dimensions. *arXiv preprint arXiv:1806.09823*, 7, 2018.

[68] Yihe Dong, Piotr Indyk, Ilya Razenshteyn, and Tal Wagner. Learning space partitions for nearest neighbor search. In *International Conference on Learning Representations*, 2019.

[69] Hao Chen, Ilaria Chillotti, Yihe Dong, Oxana Poburinnaya, Ilya Razenshteyn, and M Sadegh Riazi. $\{$SANNS$\}$: Scaling up secure approximate k-nearest neighbors search. In *29th $\{$USENIX$\}$ Security Symposium ($\{$USENIX$\}$ Security 20)*, pages 2111–2128, 2020.

[70] Jerry D Gibson and Khalid Sayood. Lattice quantization. *Advances in electronics and electron physics*, 72:259–330, 1988.

[71] William B Johnson and Joram Lindenstrauss. Extensions of lipschitz mappings into a hilbert space. *Contemporary mathematics*, 26(189-206):1, 1984.

[72] Lijie Chen. On the hardness of approximate and exact (bichromatic) maximum inner product. In *33rd Computational Complexity Conference (CCC)*, 2018.

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes] We make our contributions clear in abstract and introduction
   (b) Did you describe the limitations of your work? [Yes] We discuss the trade-off between preprocessing time and query time in Section 3
   (c) Did you discuss any potential negative societal impacts of your work? [N/A] There are no negative societal impacts of the work.
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes] We follow the ethics review guidelines and ensured that our paper conforms to them
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [Yes] We state the full assumptions of theoretical results.
   (b) Did you include complete proofs of all theoretical results? [Yes] We include complete proofs of all results.
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [N/A]
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [N/A]
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [N/A]
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [N/A]
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [N/A]
   (b) Did you mention the license of the assets? [N/A]
   (c) Did you include any new assets either in the supplemental material or as a URL? [N/A]
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A] These are openly available to use as mentioned on their website.
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A] It does not have any personal information
5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]

Appendix

Contents

Appendix

We provide supplementary materials for our work. Section A introduces the preliminary notations and definitions, Section B introduces the LSH data structure in detail for MaxIP, Section C presents our sublinear Frank-Wolfe algorithm, Section D presents the convergence analysis for sublinear Frank-Wolfe, Section E provide the algorithm and analysis on sublinear cost Herding algorithm, Section F provide the algorithm and analysis on sublinear cost policy gradient approach, Section G shows how to handle adaptive queries in MaxIP.

A Preliminary

A.1 Notations

We use $\Pr[]$ and $\mathbb{E}[]$ for probability and expectation. We use $\max\{a,b\}$ to denote the maximum between a and b. We use $\min\{a,b\}$ (resp. $\max\{a,b\}$) to denote the minimum (reps. maximum) between a and b. For a vector x, we use $\|x\|_2 := (\sum_{i=1}^n x_i^2)^{1/2}$ to denote its ℓ_2 norm. We use $\|x\|_p := (\sum_{i=1}^n |x_i|^p)^{1/p}$ to denote ℓ_p norm. For a square matrix A, we use $\mathrm{tr}[A]$ to denote the trace of matrix A.

A.2 LSH and MaxIP

We start with the defining the Approximate Nearest Neighbor (ANN) problem [61, 46, 47, 62, 63, 64, 65, 66, 67, 68, 69] as:

Definition A.1 (Approximate Nearest Neighbor (ANN))**.** *Let $\overline{c} > 1$ and $r \in (0,2)$. Given an n-vector set $P \subset \mathbb{S}^{d-1}$ on the sphere, the goal of the $(\overline{c}, r)$-Approximate Near Neighbor (ANN) search is to construct a data structure that for any query $q \in \mathbb{S}^{d-1}$ such that $\min_{p\in P} \|p-q\|_2 \leq r$, return a vector $p' \in P$ such that $\|p'-q\|_2 \leq \overline{c} \cdot r$.*

The ANN problem can be solved via locality sensitive hashing (LSH) [46, 47, 65]. In this paper, we use the standard definitions of LSH (see Indyk and Motwani [46]).

Definition A.2 (Locality Sensitive Hashing)**.** *Let $\overline{c}$ denote a parameter such that $\overline{c} > 1$. Let p_1, p_2 denote two parameters such that $0 < p_2 < p_1 < 1$. A family $\mathcal{H}$ is called $(r, \overline{c} \cdot r, p_1, p_2)$-sensitive if and only if, for any two vector $x, y \in \mathbb{R}^d$, h chosen uniformly from $\mathcal{H}$ satisfies the following:*

- *if $\|x-y\|_2 \leq r$, then $\Pr_{h\sim\mathcal{H}}[h(x) = h(y)] \geq p_1$,*
- *if $\|x-y\|_2 \geq \overline{c} \cdot r$, then $\Pr_{h\sim\mathcal{H}}[h(x) = h(y)] \leq p_2$.*

Next, we show that LSH solves ANN problem with sublinear query time complexity.

Theorem A.3 (Andoni, Laarhoven, Razenshteyn and Waingarten [66])**.** *Let $\overline{c} > 1$ and $r \in (0,2)$. The $(\overline{c}, r)$-ANN on a unit sphere $\mathbb{S}^{d-1}$ can be solved with query time $O(d \cdot n^\rho)$, space $O(n^{1+o(1)} + dn)$ and preprocessing time $O(dn^{1+o(1)})$, where $\rho = \frac{2}{\overline{c}^2} - \frac{1}{\overline{c}^4} + o(1)$.*

Here we write $o(1)$ is equivalent to $O(1/\sqrt{\log n})$. Note that we could reduce d to $n^{o(1)}$ with Johnson–Lindenstrauss Lemma [71]. Besides, we could achieve better ρ using LSH in [64] if we allowed to have more proprocessing time.

In this work, we focus on a well-known problem in computational complexity: approximate MaxIP. In this work, we follow the standard notation in [72] and define the approximate MaxIP problem as follows:

Definition A.4 (Approximate MaxIP)**.** *Let $c \in (0,1)$ and $\tau \in (0,1)$. Given an n-vector dataset $Y \subset \mathbb{S}^{d-1}$, the goal of the (c,τ)-MaxIP is to construct a data structure that, given a query $x \in \mathbb{S}^{d-1}$ with the promise that $\max_{y\in Y}\langle x, y\rangle \geq \tau$, it retrieves a vector $z \in Y$ with $\langle x, z\rangle \geq c \cdot \max_{y\in Y}\langle x, y\rangle$.*

In many applications, it is more convenient to doing inner product search in a transformed/projected space compared to doing inner product search in the original space. Thus, we propose the following definitions (Definition A.5 and Definition A.6)

Definition A.5 (Projected MaxIP)**.** *Let $\phi, \psi : \mathbb{R}^d \to \mathbb{R}^k$ denotes two transforms. Given a data set $Y \subseteq \mathbb{R}^d$ and a point $x \in \mathbb{R}^d$, we define (ϕ, ψ)-MaxIP as follows:*

$$(\phi, \psi)\text{-}\mathsf{MaxIP}(x, Y) := \max_{y \in Y} \langle \phi(x), \psi(y) \rangle$$

Definition A.6 (Projected approximate MaxIP)**.** *Let $\phi, \psi : \mathbb{R}^d \to \mathbb{R}^k$ denotes two transforms. Given an n-point dataset $Y \subset \mathbb{R}^d$ so that $\psi(Y) \subset \mathbb{S}^{d-1}$, the goal of the (c, ϕ, ψ, τ)-MaxIP is to build a data structure that, given a query $x \in \mathbb{R}^d$ and $\phi(x) \in \mathbb{S}^{k-1}$ with the promise that $\max_{y \in Y} \langle \phi(x), \psi(y) \rangle \geq \tau$, it retrieves a vector $z \in Y$ with $\langle \phi(x), \psi(z) \rangle \geq c \cdot (\phi, \psi)$-$\mathsf{MaxIP}(x, Y)$.*

Besides MaxIP, We also define a version of the minimum inner product search problem.

Definition A.7 (regularized $\mathsf{Min\text{-}IP}$)**.** *Given a data set $Y \subseteq \mathbb{R}^d$ and a point $x \in \mathbb{R}^d$. Let $\phi : \mathbb{R}^d \to \mathbb{R}^d$ denotes a mapping. Given a constant α, we define regularized $\mathsf{Min\text{-}IP}$ as follows:*

$$(\phi, \alpha)\text{-}\mathsf{Min\text{-}IP}(x, Y) := \min_{y \in Y} \langle y - x, \phi(x) \rangle + \alpha \|x - y\|.$$

A.3 Definitions and Properties for Optimization

We start with listing definitions for optimization.

Definition A.8 (Convex hull and its diameter)**.** *Given a set $A = \{x_i\}_{i \in [n]} \subset \mathbb{R}^d$, we define its convex hull $\mathcal{B}(A)$ to be the collection of all finite linear combinations y that satisfies $y = \sum_{i \in [n]} a_i \cdot x_i$ where $a_i \in (0, 1)$ for all $i \in [n]$ and $\sum_{i \in [n]} a_i = 1$. Let $D_{\max}$ denotes the maximum square of diameter of $\mathcal{B}(A)$ so that $\|x - y\|_2 \leq D_{\max}$ for all $(x, y) \in \mathcal{B}(A)$.*

Definition A.9 (Smoothness)**.** *We say L is β-smooth if*

$$L(y) \leq L(x) + \langle \nabla L(x), y - x \rangle + \frac{\beta}{2} \|y - x\|_2^2$$

Definition A.10 (Convex)**.** *We say function L is convex if*

$$L(x) \geq L(y) + \langle \nabla L(y), x - y \rangle$$

Next, we list properties for optimization.

Corollary A.11. *For a set $A = \{x_i\}_{i \in [n]} \subset \mathbb{R}^d$, and its convex hull $\mathcal{B}(A)$, given a query $q \in \mathbb{R}^d$, if $x^* = \arg\max_{x \in A} q^\top x$. Then, $q^\top y \leq q^\top x^*$ for all $y \in \mathcal{B}(A)$.*

Proof. We can upper bound $q^\top y$ as follows:

$$\begin{aligned} q^\top y &= q^\top (\sum_{i \in [n]} a_i \cdot x_i) \\ &= \sum_{i \in [n]} a_i \cdot q^\top x_i \\ &\leq \sum_{i \in [n]} a_i \cdot q^\top x^* \\ &\leq q^\top x^* \end{aligned}$$

where the first step follows from the definition of convex hull in Definition A.8, the second step is an reorganization, the third step follows the fact that $a_i \in [0, 1]$ for all $i \in [n]$ and $q^\top x_i \leq q^\top x^*$ for all $x_i \in A$, the last step follows that $\sum_{i \in [n]} a_i \leq 1$. $\square$

Lemma A.12 (MaxIP Condition)**.** *Let $g : \mathbb{R}^d \to \mathbb{R}$ denotes a convex function. Let $S \subset \mathbb{R}^d$ denotes a set of points. Given a vector $x \in \mathcal{B}(S)$, we have*

$$\min_{s \in S} \langle \nabla g(x), s - x \rangle \leq 0, \quad \forall x \in \mathcal{B}.$$

Proof. Let $s_{\min} = \arg\min_{s \in S} \langle \nabla g(x), s \rangle$. Then, we upper bound $\langle \nabla g(x), s_{\min} - x \rangle$ as

$$\begin{aligned}
\langle \nabla g(x), s_{\min} - x \rangle &= \langle \nabla g(x), s_{\min} - \sum_{s \in S} a_i \cdot s \rangle \\
&\leq \langle \nabla g(x), \sum_{s \in S} a_i (s_{\min} - s_i) \rangle \\
&= \sum_{s_i \in S} a_i \langle \nabla g(x), s_{\min} - s_i \rangle \\
&= \sum_{s_i \in S} a_i (\langle \nabla g(x), s_{\min} \rangle - \langle \nabla g(x), s_i \rangle) \\
&\leq 0 \qquad (9)
\end{aligned}$$

where the first step follows from the definition of convex hull in Definition A.8, the second and third steps are reorganizations, the final steps follows that $\langle \nabla g(x), s_0 \rangle \leq \langle \nabla g(x), s \rangle$ for all $s \in S$.

Next, we upper bound $\min_{s \in S} \langle \nabla g(x), s - x \rangle \leq 0, \quad \forall x \in \mathcal{B}$ as

$$\min_{s \in S} \langle \nabla g(x), s - x \rangle \leq \langle \nabla g(x), s_0 - x \rangle \leq 0$$

where the first step follows from the definition of function min and the second step follows from Eq (9). □

B Data Structures

In this section, we present a formal statement that solves (c, τ)-MaxIP problem on unit sphere using LSH for $(\overline{c}, r)$-ANN.

Corollary B.1 (Formal statement of Corollary 4.4)**.** *Let $c \in (0,1)$ and $\tau \in (0,1)$. Given a set of n-vector set $Y \subset \mathcal{S}^{d-1}$ on the sphere, one can construct a data structure with $O(dn^{1+o(1)})$ preprocessing time and $O(n^{1+o(1)} + dn)$ space so that for any query $x \in \mathcal{S}^{d-1}$, we take query time complexity $O(d \cdot n^{\rho})$ to retrieve (c, τ)-MaxIP of x in Y with probability at least 0.9[i], where $\rho := \frac{2(1-\tau)^2}{(1-c\tau)^2} - \frac{(1-\tau)^4}{(1-c\tau)^4} + o(1)$*

Proof. We know that $\|x - y\|_2^2 = 2 - 2\langle x, y \rangle$ for all $x, y \in \mathcal{S}^{d-1}$. In this way, if we have a LSH data-structure for $(\overline{c}, r)$-ANN. It could be used to solve (c, τ)-MaxIP with $\tau = 1 - 0.5r^2$ and $c = \frac{1 - 0.5\overline{c}^2 r^2}{1 - 0.5r^2}$. Next, we write $\overline{c}^2$ as

$$\overline{c}^2 = \frac{1 - c(1 - 0.5r^2)}{0.5r^2} = \frac{1 - c\tau}{1 - \tau}.$$

Next, we show that if the LSH is initialized following Theorem A.3, it takes query time $O(d \cdot n^{\rho})$, space $O(n^{1+o(1)} + dn)$ and preprocessing time $O(dn^{1+o(1)})$ to solve (c, τ)-MaxIP through solving $(\overline{c}, r)$-ANN, where

$$\rho = \frac{2}{\overline{c}^2} - \frac{1}{\overline{c}^4} + o(1) = \frac{2(1-\tau)^2}{(1-c\tau)^2} - \frac{(1-\tau)^4}{(1-c\tau)^4} + o(1).$$

□

In practice, c is increasing as we set parameter τ close to $\mathrm{MaxIP}(x, Y)$. Moreover, Corrolary B.1 could be applied to projected MaxIP problem.

Corollary B.2. *Let $c \in (0,1)$ and $\tau \in (0,1)$. Let $\phi, \psi : \mathbb{R}^d \to \mathbb{R}^k$ denotes two transforms. Let $\mathcal{T}_\phi$ denotes the time to compute $\phi(x)$ and $\mathcal{T}_\psi$ denotes the time to compute $\psi(y)$. Given a set of n-points $Y \in \mathbb{R}^d$ with $\psi(Y) \subset \mathcal{S}^{k-1}$ on the sphere, one can construct a data structure with*

[i]It is obvious to boost probability from constant to δ by repeating the data structure $\log(1/\delta)$ times.

*$O(dn^{1+o(1)} + \mathcal{T}_\psi n)$ preprocessing time and $O(n^{1+o(1)} + dn)$ space so that for any query $x \in \mathbb{R}^d$ with $\phi(x) \in \mathcal{S}^{k-1}$, we take query time complexity $O(d \cdot n^\rho + \mathcal{T}_\phi)$ to solve (c, ϕ, ψ, τ)-*MaxIP *with respect to (x, Y) with probability at least 0.9, where $\rho := \frac{2(1-\tau)^2}{(1-c\tau)^2} - \frac{(1-\tau)^4}{(1-c\tau)^4} + o(1)$.*

Proof. The preprocessing phase can be decomposed in two parts.

- It takes $O(\mathcal{T}_\psi n)$ time to transform every $y \in Y$ into $\psi(y)$.
- It takes $O(O(dn^{1+o(1)})$ time and $O(dn^{1+o(1)} + dn)$ to index every $\psi(y)$ into LSH using Corrolary B.1.

The query phase can be decomposed in two parts.

- It takes $O(\mathcal{T}_\phi)$ time to transform every $x \in \mathbb{R}^d$ into $\phi(x)$.
- It takes $O(d \cdot n^\rho)$ time perform query for $\phi(x)$ in LSH using Corrolary B.1.

□

C Algorithms

C.1 Problem Formulation

In this section, we show how to use Frank-Wolfe Algorithm to solve the Problem C.1.

Problem C.1.

$$\min_{w \in \mathcal{B}} g(w) \tag{10}$$

We have the following assumptions:

- *$g : \mathbb{R}^d \to \mathbb{R}$ is a differentiable function.*
- *$S \subset \mathbb{R}^d$ is a finite feasible set. $|S| = n$.*
- *$\mathcal{B} = \mathcal{B}(S) \subset \mathbb{R}^d$ is the convex hull of the finite set $S \subset \mathbb{R}^d$ defined in Definition A.8.*
- *$D_{\max}$ is the maximum diameter of $\mathcal{B}(S)$ defined in Definition A.8*

In Problem C.1, function g could have different proprieties about convexity and smoothness.

To solve this problem, we introduce a Frank-Wolfe Algorithm shown in Algorithm 1.

Algorithm 1 Frank-Wolf algorithm for Problem C.1

```
1:  procedure FRANKWOLFE($S \subset \mathbb{R}^d$)
2:      $T \leftarrow O(\frac{\beta D_{\max}^2}{\epsilon}), \forall t \in [T]$
3:      $\eta \leftarrow \frac{2}{t+2}$
4:      Start with $w^0 \in \mathcal{B}$.                ▷ $\mathcal{B} = \mathcal{B}(S)$(see Definition A.8).
5:      for $t = 1 \to T - 1$ do
6:          $s^t \leftarrow \arg\min_{s \in S} \langle \nabla g(w^t), s \rangle$
7:          $w^{t+1} \leftarrow (1 - \eta_t) w^t + \eta_t s^t$
8:      end for
9:      return $w^T$
10: end procedure
```

One of the major computational bottleneck of Algorithm 1 is the cost paid in each iteration. Algorithm 1 has to linear scan all the $s \in S$ in each iteration. To tackle this issue, we propose a Frank-Wolfe Algorithm with sublinear cost in each iteration.

C.2 Our Sublinear Frank-Wolfe Algorithm

In this section, we present the Frank-Wolfe algorithm with sublinear cost per iteration using LSH. The first step is to formulate the line 6 in Algorithm 1 as a projected MaxIP problem defined in Definition A.5. To achieve this, we present a general MaxIP transform.

Proposition C.2 (MaxIP Transform)**.** *Let $\phi_1, \psi_1 : \mathbb{R}^d \to \mathbb{R}^{k_1}$ and $\phi_2, \psi_2 : \mathbb{R}^d \to \mathbb{R}^{k_2}$ to be the projection functions. Given the polynomial function $p(z) = \sum_{i=0}^{D} a_i z^i$, we show that*

$$\langle \phi_1(x), \psi_1(y) \rangle + p(\|\phi_2(x) - \psi_2(y)\|_2^2) = \langle \phi(x), \psi(y) \rangle \tag{11}$$

where $\phi, \psi : \mathbb{R}^d \to \mathbb{R}^{k_1 + k_2(D+1)^2}$ is the decomposition function.

Proof. Because $\phi_2(x), \psi_2(y) \in \mathbb{R}^{k_2}$, $\|\phi_2(x) - \psi_2(y)\|_2^{2i} = \sum_{j=1}^{k_2} (\phi_2(x)_j - \psi_2(y)_j)^{2i}$. This is the sum over dimensions. Then, we have

$$\begin{aligned} p(\|\phi_2(x) - \psi_2(y)\|_2^2) &= \sum_{i=0}^{D} a_i \|\phi_2(x) - \psi_2(y)\|_2^{2i} \\ &= \sum_{i=0}^{D} a_i \sum_{j=1}^{k_2} (\phi_2(x)_j - \psi_2(y)_j)^{2i} \end{aligned}$$

where the first follows from definition of polynomial p, and the second step follows from definition of ℓ_2 norm.

Here $\phi_2(x)_j$ means the jth entry of $\phi_2(x)$. Using the binomial theorem, we decompose $(\phi_2(x)_j - \psi_2(y)_j)^{2i}$ as:

$$\begin{aligned} & (\phi_2(x)_j - \psi_2(y)_j)^{2i} \\ = & \sum_{l=0}^{2i} \binom{2i}{l} \phi_2(x)_j^{2i-l} \psi_2(y)_j^{l} \\ = & \langle \underbrace{[\phi_2(x)_j^{2i}, \cdots, \phi_2(x)_j^{2i-l}, \cdots, \phi_2(x)_j, 1]}_{u_j}, \underbrace{[1, \psi_2(y)_j, \cdots, \psi_2(y)_j^{l}, \cdots, \psi_2(y)_j^{2i}]}_{v_j} \rangle \end{aligned}$$

Then, we generate two vectors $u^i \in \mathbb{R}^{k_2(2i+1)}$ and $v^i \in \mathbb{R}^{k_2(2i+1)}$

$$\begin{aligned} u^i &= \begin{bmatrix} u_1 & \cdots & u_j & \cdots & u_{k_2} \end{bmatrix} & u_j &= \begin{bmatrix} \phi_2(x)_j^{2i} & \cdots & \phi_2(x)_j^{2i-l} & \cdots & \phi_2(x)_j & 1 \end{bmatrix}^\top \\ v^i &= \begin{bmatrix} v_1 & \cdots & v_j & \cdots & v_{k_2} \end{bmatrix} & v_j &= \begin{bmatrix} 1 & \psi_2(y)_j & \cdots & \psi_2(y)_j^{l} & \cdots & \psi_2(y)_j^{2i} \end{bmatrix}^\top \end{aligned}$$

Thus, $\sum_{j=1}^{k_2} (\phi_2(x)_j - \psi_2(y)_j)^{2i}$ can be rewrite with inner product by concatenating all the u_j together and then concatenating all the v_j.

$$\sum_{j=1}^{k_2} (\phi_2(x)_j - \psi_2(y)_j)^{2i} = \langle u^i, v^i \rangle.$$

We make vectors $b \in \mathbb{R}^{k_2(D+1)^2}$ and $c \in \mathbb{R}^{k_2(D+1)^2}$ such as

$$\begin{aligned} b &= [u^0 \cdots, u^i, \cdots, u_D] \\ c &= [a_0 v^0, \cdots, a_i v^i, \cdots, a_D v^D] \end{aligned}$$

So that

$$\sum_{i=0}^{D} a_i \sum_{j=1}^{k_2} (\phi_2(x)_j - \psi_2(y)_j)^{2i} = \sum_{i=0}^{D} a_i \langle u^i, v^i \rangle = \sum_{i=0}^{D} \langle u^i, a_i v^i \rangle = \langle b, c \rangle$$

Finally, we have

$$\begin{aligned}\langle \phi_1(x), \psi_1(y)\rangle + p(\|\phi_2(x) - \psi_2(y)\|_2^2) &= \langle \phi_1(x), \psi_1(y)\rangle + \langle b, c\rangle \\ &= \langle [\phi_1(x), b], [\psi_1(y), c]\rangle \\ &= \langle \phi(x), \psi(y)\rangle\end{aligned}$$

Total projected dimension:

$$\begin{aligned}k_1 + \sum_{i=0}^{D} k_2(2i+1) &= k_1 + (D+1)k_2 + 2k_2\sum_{i=1}^{D} i \\ &= k_1 + (D+1)k_2 + 2k_2 \cdot \frac{D(D+1)}{2} \\ &= k_1 + k_2(D+1)^2\end{aligned}$$

□

Therefore, any binary function with format $\langle \phi_1(x), \psi_1(y)\rangle + p(\|\phi_2(x) - \psi_2(y)\|_2^2)$ defined in Proposition C.2 can be transformed as a inner product.

Next, we show that a modified version of line 6 in Algorithm 1 can be formulated as a projected MaxIP problem.

Corollary C.3 (Equivalence between projected MaxIP and Min-IP)**.** *Let g be a differential function defined on convex set $\mathcal{K} \subset \mathbb{R}^d$. Given $\eta \in (0,1)$ and $x, y \in \mathcal{K}$, we define $\phi, \psi : \mathbb{R}^d \to \mathbb{R}^{d+3}$ as follows:*

$$\phi(x) := \begin{bmatrix} \frac{\phi_0(x)^\top}{D_x} & 0 & \sqrt{1 - \frac{\|\phi_0(x)\|_2^2}{D_x^2}} \end{bmatrix}^\top \quad \psi(y) := \begin{bmatrix} \frac{\psi_0(y)^\top}{D_y} & \sqrt{1 - \frac{\|\psi_0(y)\|_2^2}{D_y^2}} & 0 \end{bmatrix}^\top$$

where

$$\phi_0(x) := [\nabla g(x)^\top, x^\top \nabla g(x)]^\top \quad \psi_0(y) := [-y^\top, 1]^\top$$

, D_x is the maximum diameter of $\phi_0(x)$ and D_y is the maximum diameter of $\psi_0(y)$.

Then, for all $x, y \in \mathbb{R}^d$, we transform them into unit vector $\phi(x)$ and $\psi(y)$ on $\mathcal{S}^{d+2}$. Moreover, we have

$$\langle y - x, \nabla g(x)\rangle = -D_x D_y \langle \phi(x), \psi(y)\rangle$$

Further, the (ϕ, ψ)-MaxIP (Definition A.5) is equivalent to the $(\nabla g, 0)$-Min-IP (Definition A.7).

$$\arg\max_{y \in \mathcal{K}} \langle \phi(x), \psi(y)\rangle = \arg\min_{y \in \mathcal{K}} \langle y - x, \nabla g(x)\rangle$$

In addition, let $\mathcal{T}_\psi$ denote the time of evaluating at any point $y \in \mathbb{R}^d$ for function ψ, then we have $\mathcal{T}_\psi = O(1)$.

Let $\mathcal{T}_\phi$ denote the time of evaluating at any point $x \in \mathbb{R}^d$ for function ϕ, then we have $\mathcal{T}_\phi = \mathcal{T}_{\nabla g} + O(d)$, where the $\mathcal{T}_{\nabla g}$ denote the time of evaluating function ∇g at any point $x \in \mathbb{R}^d$.

Proof. We start with showing that $\|\phi(x)\|_2 = \|\psi(y)\|_2 = 1$. Next, we show that

$$\begin{aligned}\langle \phi(x), \psi(y)\rangle &= \frac{\langle \phi_0(x), \psi_0(y)\rangle}{D_x D_y} \\ &= \frac{\langle -y, \nabla g(x)\rangle + \langle x, \nabla g(x)\rangle}{D_x D_y} \\ &= -\frac{\langle y - x, \nabla g(x)\rangle}{D_x D_y}\end{aligned}$$

where the first step follows from definition of ϕ and ψ, the second step follows from definition of ϕ_0 and ψ_0, the last step is a reorganization.

Based on the results above,

$$\arg\max_{y \in \mathcal{K}} \langle \phi(x), \psi(y)\rangle = \arg\min_{y \in \mathcal{K}} \langle y - x, \nabla g(x)\rangle$$

□

Using Corollary C.3, the direction search in Frank-Wolfe algorithm iteration is equivalent to a (ϕ, ψ)-MaxIP problem. In this way, we propose Algorithm 2, an Frank-Wolfe algorithm with sublinear cost per iteration using LSH.

Algorithm 2 Sublinear Frank-Wolfe for Problem C.1

```
1: data structure LSH                                                    ▷ Corollary B.2
2:     INIT($S \subset \mathbb{R}^d$, $n \in \mathbb{N}$, $d \in \mathbb{N}$, $c \in (0,1)$)
3:         ▷ $|S| = n$, $c \in (0,1)$ is LSH parameter, and $d$ is the dimension of data
4:     QUERY($x \in \mathbb{R}^d$, $\tau \in (0,1)$)                    ▷ $\tau \in (0,1)$ is LSH parameter
5: end data structure
6:
7: procedure SUBLINEARFRANKWOLFE($S \subset \mathbb{R}^d$, $n \in \mathbb{N}$, $d \in \mathbb{N}$, $c \in (0,1)$, $\tau \in (0,1)$)   ▷ Theorem D.1
8:     Construct $\phi, \psi : \mathbb{R}^d \to \mathbb{R}^{d+1}$ as Corollary C.3
9:     static LSH LSH
10:    LSH.INIT($\psi(S), n, d+3, c$)
11:    Start with $w^0 \in \mathcal{B}$.                               ▷ $\mathcal{B} = \mathcal{B}(S)$(see Definition A.8).
12:    $T \leftarrow O(\frac{\beta D_{\max}^2}{c^2 \epsilon})$
13:    $\eta \leftarrow \frac{2}{c(t+2)}, \forall t \in [T]$
14:    for $t = 1 \to T-1$ do
15:        /* Query with $w^t$ and retrieve its $(c, \phi, \psi, \tau)$-MaxIP $s^t \in S$ from LSH data structure */
16:        $s^t \leftarrow$ LSH.QUERY($\phi(w^t), \tau$)
17:        /* Update $w^t$ in the chosen direction*/
18:        $w^{t+1} \leftarrow (1-\eta_t) \cdot w^t + \eta_t \cdot s^t$
19:    end for
20:    return $w^T$
21: end procedure
```

D Convergence Analysis

In this Section D, analyze the convergence of our Sublinear Frank-Wolfe algorithm in Algorithm 2 when g is convex (see Definition A.10) and β-smooth (see Definition A.9). Moreover, we compare our sublinear Frank-Wolfe algorithm with Frank-Wolfe algorithm in Algorithm 1 in terms of number of iterations and cost per iteration.

D.1 Summary

We first show the comparsion results in Table 2. As shown in the table, with $O(dn^{1+o(1)} \cdot \kappa)$ preprocessing time, Algorithm 2 achieves $O(dn^\rho \cdot \kappa + \mathcal{T}_g)$ cost per iteration with $\frac{1}{c^2}$ more iterations.

Algorithm	**Statement**	**Preprocessing**	**#iters**	**cost per iter**
Algorithm 1	[9]	0	$O(\beta D_{\max}^2/\epsilon)$	$O(dn + \mathcal{T}_g)$
Algorithm 2	Theorem D.1	$O(dn^{1+o(1)} \cdot \kappa)$	$O(c^{-2}\beta D_{\max}^2/\epsilon)$	$O(dn^\rho \cdot \kappa + \mathcal{T}_g)$

Table 2: Comparison between original Frank-Wolfe algorithm and our sublinear Frank-Wolfe algorithm. Here $\mathcal{T}_g$ denotes the time for computing gradient of g, $c \in (0,1)$ is the approximation factor of LSH. We let $\kappa := \Theta(\log(T/\delta))$ where T is the number of iterations and δ is the failure probability. $\rho \in (0,1)$ is a fixed parameter determined by LSH.

D.2 Sublinear Frank-Wolfe Algorithm

The goal of this section is to prove Theorem D.1.

Theorem D.1 (Convergence result of Sublinear Frank-Wolfe, a formal version of Theorem 3.1)**.** *Let $g : \mathbb{R}^d \to \mathbb{R}$ denotes a convex (see Definition A.10) and β-smooth function (see Definition A.9).*

Let the complexity of calculating $\nabla g(x)$ to be $\mathcal{T}_g$. Let $\phi, \psi : \mathbb{R}^d \to \mathbb{R}^k$ denotes two transforms in Corollary C.3. Let $S \subset \mathbb{R}^d$ denotes a set of points with $|S| = n$, and $\mathcal{B} \subset \mathbb{R}^d$ is the convex hull of S (see Definition A.8). For any parameters ϵ, δ, there is an iterative algorithm with that takes $O(dn^{1+o(1)} \cdot \kappa)$ preprocessing time and $O((n^{1+o(1)} + dn) \cdot \kappa)$ space, takes $T = O(\frac{\beta D_{\max}^2}{\epsilon})$ iterations and $O(dn^{\rho} \cdot \kappa + \mathcal{T}_g)$ cost per iteration, starts from a random w^0 from $\mathcal{B}$ as initialization point, updates the w in each iteration as follows:

$$s^t \leftarrow (c, \phi, \psi, \tau)\text{-}\mathsf{MaxIP} \textit{ of } w^t \textit{ with respect to } S$$
$$w^{t+1} \leftarrow w^t + \eta \cdot (s^t - w^t)$$

and outputs $w^T \in \mathbb{R}^d$ from $\mathcal{B}$ such that

$$g(w^T) - \min_{w \in \mathcal{B}} g(w) \leq \epsilon,$$

holds with probability at least $1 - \delta$. Here $\kappa := \Theta(\log(T/\delta))$ and $\rho := \frac{2(1-\tau)^2}{(1-c\tau)^2} - \frac{(1-\tau)^4}{(1-c\tau)^4} + o(1)$.

Proof. **Convergence.**

Let t denote some fixed iteration. We consider two cases:

- **Case 1.** $\tau > \max_{s \in S} \langle \psi(s), \phi(w^t) \rangle$;
- **Case 2.** $\tau \leq \max_{s \in S} \langle \psi(s), \phi(w^t) \rangle$.

Case 1. In this case, we can show that

$$\begin{aligned} \tau &\geq \max_{s \in S} \langle \psi(s), \phi(w^t) \rangle \\ &\geq \frac{\langle \psi(w^*), \phi(w^t) \rangle}{D_x D_y} \\ &= \frac{\langle w^t - w^*, \nabla g(w^t) \rangle}{D_x D_y} \\ &\geq \frac{g(w^t) - g(w^*)}{D_x D_y}, \end{aligned}$$

where the first step follows from Corollary C.3, the second step follows from the Corollary A.11, the third step is a reorganization, the last step follows the convexity of g (see Definition A.10).

Thus, as long as $\tau \geq D_x D_y \epsilon$, then we have

$$g(w^t) - g(w^*) \leq \epsilon.$$

This means we already converges to the optimal.

Case 2. We start with the upper bounding $\langle s^t - w^t, \nabla g(w^t) \rangle$ as

$$\begin{aligned} \langle s^t - w^t, \nabla g(w^t) \rangle &= -D_x D_y \langle \psi(s^t), \phi(w^t) \rangle \\ &\leq -c \cdot D_x D_y \max_{s \in S} \langle \psi(s), \phi(w^t) \rangle \\ &\leq -c \cdot D_x D_y \langle \psi(w^*), \phi(w^t) \rangle \\ &= c \langle w^* - w^t, \nabla g(w^t) \rangle \end{aligned} \tag{12}$$

where the first step follows from Corollary C.3, the second step follows from Corollary B.2 and MaxIP condition in Lemma A.12, the third step follows from Corollary A.11.

For convenient of the proof, for each t, we define h_t as follows:

$$h_t = g(w^t) - g(w^*). \tag{13}$$

Next, we upper bound h_{t+1} as

$$\begin{aligned}
h_{t+1} &= g(w^{t+1}) - g(w^*) \\
&= g((1-\eta_t)w^t + \eta_t s^t) - g(w^*) \\
&\leq g(w^t) + \eta_t \langle s^t - w^t, \nabla g(w^t)\rangle + \frac{\beta}{2}\eta_t^2 \|s^t - w^t\|_2^2 - g(w^*) \\
&\leq g(w^t) + \eta_t \langle s^t - w^t, \nabla g(w^t)\rangle + \frac{\beta D_{\max}^2}{2}\eta_t^2 - g(w^*) \\
&\leq g(w^t) + c\eta_t \langle w^* - w^t, \nabla g(w^t)\rangle + \frac{\beta D_{\max}^2}{2}\eta_t^2 - g(w^*) \\
&= (1-\eta_t)g(w^t) + c\eta_t \left(g(w^t) + \langle w^* - w^t, \nabla g(w^t)\rangle\right) + \frac{\beta D_{\max}^2}{2}\eta_t^2 - g(w^*) \\
&\leq (1-\eta_t)g(w^t) + c\eta_t g(w^*) + \frac{\beta D_{\max}^2}{2}\eta_t^2 - g(w^*) \\
&\leq (1-c\eta_t)g(w^t) - (1-c\eta_t)g(w^*) + \frac{\beta D_{\max}^2}{2}\eta_t^2 \\
&\leq (1-c\eta_t)h_t + \frac{\beta D_{\max}^2}{2}\eta_t^2
\end{aligned} \tag{14}$$

where the first step follows from definition of h_{t+1} (see Eq. (13)), the second step follows from the update rule of Frank-Wolfe, the third step follows from the definition of β-smoothness in Definition A.9, the forth step follows from the definition of maximum diameter in Definition A.8, the fifth step follows the Eq (12), the sixth step is a reorganization, the seventh step follows from the definition of convexity (see Definition A.10), the eighth step follows from merging the coefficient of $g(w^*)$, and the last step follows from definition of h_t (see Eq. (13)).

Let $e_t = A_t h_t$, A_t is a parameter and we will decide it later. we have:

$$\begin{aligned}
e_{t+1} - e_t &= A_{t+1}\left((1-c\eta_t)h_t + \frac{\beta D_{\max}^2}{2}\eta_t^2\right) - A_t h_t \\
&= \left(A_{t+1}(1-c\eta_t) - A_t\right)h_t + \sigma + \frac{\beta D_{\max}^2}{2}A_{t+1}\eta_t^2
\end{aligned} \tag{15}$$

Let $A_t = \frac{t(t+1)}{2}$, $c\eta_t = \frac{2}{t+2}$. In this way we rewrite $A_{t+1}(1-\eta_t) - A_t$ and $A_{t+1}\frac{\eta_t^2}{2}$ as

- $A_{t+1}(1-\eta_t) - A_t = 0$
- $A_{t+1}\frac{\eta_t^2}{2} = \frac{t+1}{(t+2)c^2} < c^{-2}$

Next, we upper bound $e_{t+1} - e_t$ as:

$$\begin{aligned}
e_{t+1} - e_t &< 0 + c^{-2}\frac{t+1}{t+2}\beta D_{\max}^2 \\
&< c^{-2}\beta D_{\max}^2
\end{aligned} \tag{16}$$

where the first step follows from $A_{t+1}(1-\eta_t) - A_t = 0$ and $A_{t+1}\frac{\eta_t^2}{2} = \frac{t+1}{(t+2)c^2}$. The second step follows from $\frac{t+1}{t+2} < 1$

Based on Eq (16), we upper bound e_t using induction and have

$$e_t < c^{-2}t\beta D_{\max}^2 \tag{17}$$

Using the definition of e_t, we have

$$h_t = \frac{e_t}{A_t} < \frac{2\beta D_{\max}^2}{c^2(t+1)} \tag{18}$$

To make $h_t \leq \epsilon$, t should be in $O(\frac{\beta D_{\max}^2}{c^2 \epsilon})$. Thus, we complete the proof.

Preprocessing time According to Corrollary B.2, can construct $\kappa = \Theta(\log(T/\delta))$ LSH data structures for (c, ϕ, ψ, τ)-MaxIP with ϕ, ψ defined in Corollary C.3. As transforming every $s \in S$ into $\psi(s)$ takes $O(dn)$. Therefore, the total the preprocessing time complexity is $O(dn^{1+o(1)} \cdot \kappa)$.

Cost per iteration

Given each w^t, compute $\nabla g(w^t)$ takes $\mathcal{T}_g$. Next, it takes $O(d)$ time to generate $\phi(w^t)$ according to Corollary C.3 based on $g(w^t)$ and $\nabla g(w^t)$. Next, according to Corrollary B.2, it takes $O(dn^\rho \cdot \kappa)$ to retrieve s^t from $\kappa = \Theta(\log(T/\delta))$ LSH data structures. After we select s^t, it takes $O(d)$ time to update the w^{t+1}. Combining the time for gradient calculation, LSH query and w^t update, the total complexity is $O(dn^\rho \cdot \kappa + \mathcal{T}_g)$ with $\rho := \frac{2(1-\tau)^2}{(1-c\tau)^2} - \frac{(1-\tau)^4}{(1-c\tau)^4} + o(1)$.

$\square$

E Herding Algorithm

E.1 Problem Formulation

In this section, we focus on herding algorithm a specific example of Problem C.1. We consider a finite set $\mathcal{X} \subset \mathbb{R}^d$ and a mapping $\Phi : \mathbb{R}^d \to \mathbb{R}^k$. Given a distribution $p(x)$ over $\mathcal{X}$, we denote $\mu \in \mathbb{R}^k$ as

$$\mu = \mathop{\mathbb{E}}_{x \sim p(x)}[\Phi(x)] \tag{19}$$

The goal of herding algorithm [57] is to find T elements $\{x_1, x_2, \cdots, x_T\} \subseteq \mathcal{X}$ such that $\|\mu - \sum_{t=1}^{T} v_t \Phi(x_t)\|_2$ is minimized. Where v_t is a non-negative weight. The algorithm generates samples by the following:

$$\begin{aligned} x_{t+1} &= \arg\max_{x \in \mathcal{X}} \langle w_t, \Phi(x) \rangle \\ w_{t+1} &= w_t + \mu - \Phi(x_{t+1}) \end{aligned} \tag{20}$$

Let $\mathcal{B}$ denotes the convex hull of X. [1] show that the recursive algorithm in Eq (20) is equivalent to a Frank-Wolfe algorithm Problem E.1.

Problem E.1 (Herding)**.**

$$\min_{w \in \mathcal{B}} \frac{1}{2}\|w - \mu\|_2^2$$

We have the following assumptions:

- *$S = \Phi(\mathcal{X}) \subset \mathbb{R}^d$ is a finite feasible set. $|S| = n$.*
- *$\mathcal{B} = \mathcal{B}(S) \subset \mathbb{R}^d$ is the convex hull of the finite set $S \subset \mathbb{R}^d$ defined in Definition A.8.*
- *$D_{\max}$ is the maximum diameter of $\mathcal{B}(S)$ defined in Definition A.8*

Therefore, a frank-Wolfe algorithm [1] for herding is proposed as

Algorithm 3 Frank-Wolf algorithm for Herding

```
1:  procedure FRANKWOLFE($S \subset \mathbb{R}^k$)
2:      $T \leftarrow O(\frac{D_{\max}^2}{\epsilon}), \forall t \in [T]$
3:      $\eta \leftarrow \frac{2}{t+2}$
4:      Start with $w^0 \in \mathcal{B}$.
5:      for $t = 1 \to T - 1$ do
6:          $s^t \leftarrow \arg\max_{s \in S} \langle w^t - \mu, s \rangle$
7:          $w^{t+1} \leftarrow (1 - \eta) w^t + \eta s^t$
8:      end for
9:      return $w^T$
10: end procedure
```

Algorithm 3 takes $O(nd)$ cost per iteration.

To improve the efficiency of Algorithm 3, we propose a herding algorithm with sublinear cost per iteration using LSH.

Algorithm 4 Sublinear Frank-Wolf algorithm for Herding

```
1:  data structure LSH                                                  ▷ Corollary B.2
2:      INIT($S \subset \mathbb{R}^d$, $n \in \mathbb{N}$, $d \in \mathbb{N}$, $c \in (0,1)$)
3:                   ▷ $|S| = n$, $c \in (0,1)$ is LSH parameter, and $d$ is the dimension of data
4:      QUERY($x \in \mathbb{R}^d$, $\tau \in (0,1)$)                       ▷ $\tau \in (0,1)$ is LSH parameter
5:  end data structure
6:
7:  procedure SUBLINEARFRANKWOLF($S \subset \mathbb{R}^d$, $n \in \mathbb{N}$, $d \in \mathbb{N}$,$c \in (0,1)$ ,$\tau \in (0,1)$ )
8:                                                                          ▷ Theorem E.3
9:      Construct $\phi, \psi : \mathbb{R}^d \to \mathbb{R}^{d+1}$ as Corollary C.3
10:     static LSH LSH
11:     LSH.INIT($\psi(S), n, d+3, c$)
12:     Start with $w^0 \in \mathcal{B}$.                     ▷ $\mathcal{B} = \mathcal{B}(S)$(see Definition A.8).
13:     $T \leftarrow O(\frac{\beta D_{\max}^2}{c^2\epsilon}), \forall t \in [T]$
14:     $\eta \leftarrow \frac{2}{c(t+2)}$
15:     for $t = 1 \to T-1$ do
16:         /* Query with $w^t$ and retrieve its $(c,\phi,\psi)$-MaxIP $s^t \in S$ from LSH data structure */
17:         $s^t \leftarrow$ LSH.QUERY($\phi(w^t), \tau$)
18:         /* Update $w^t$ in the chosen direction*/
19:         $w^{t+1} \leftarrow (1-\eta_t) \cdot w^t + \eta_t \cdot s^t$
20:     end for
21:     return $w^T$
22: end procedure
```

884 **E.2 Convergence Analysis**

885 The goal of this section is to show the convergence analysis of our Algorithm 4 compare it with
886 Algorithm 3 for herding.

We first show the comparison results in Table 3.

Algorithm	Statement	Preprocessing	#iters	cost per iter
Algorithm 3	[1]	0	$O(D_{\max}^2/\epsilon)$	$O(nd)$
Algorithm 4	Theorem E.3	$O(\kappa n^{1+o(1)})$	$O(c^{-2}D_{\max}^2/\epsilon)$	$O(\kappa n^\rho \log n + d)$

Table 3: Comparison between Algorithm 4 and Algorithm 3

887

888 Next, we analyze the smoothness of $\frac{1}{2}\|w-\mu\|_2^2$.

889 **Lemma E.2.** *We show that $g(w) = \frac{1}{2}\|w^T - \mu\|_2^2$ is a convex and 1-smooth function.*

Proof.

$$\begin{aligned} g(x) + \langle \nabla g(x), y-x\rangle + \frac{1}{2}\|y-x\|_2^2 &= \frac{1}{2}\|x-\mu\|_2^2 + \langle x-\mu, y-x\rangle + \frac{1}{2}\|y-x\|_2^2 \\ &= \frac{1}{2}(x^\top x - 2x^\top \mu + \mu^\top \mu) + (x^\top y - y^\top \mu \\ &= \frac{1}{2}y^\top y - y^\top \mu + \frac{1}{2}\mu^\top \mu \\ &= \frac{1}{2}\|y-\mu\|_2^2 \\ &= g(y) \end{aligned} \tag{21}$$

890 where all the steps except the last step are reorganizations. The last step follows $g(y) = \frac{1}{2}\|y-\mu\|_2^2$
891 Rewrite the Eq (21) above, we have

$$g(y) = g(x) + \langle \nabla g(x), y-x\rangle + \frac{1}{2}\|y-x\|_2^2 \tag{22}$$

$$\geq g(x) + \langle \nabla g(x), y - x \rangle \tag{23}$$

$g(x) = \frac{1}{2}\|x - \mu\|_2^2$ is a convex function.

Rewrite the Eq (21) above again, we have

$$g(y) = g(x) + \langle \nabla g(x), y - x \rangle + \frac{1}{2}\|y - x\|_2^2 \tag{24}$$

$$\leq g(x) + \langle \nabla g(x), y - x \rangle + \frac{1}{2}\|y - x\|_2^2 \tag{25}$$

$g(x) = \frac{1}{2}\|x - \mu\|_2^2$ is a 1-smooth convex function.

□

Next, we show the convergence results of Algorithm 4.

Theorem E.3 (Convergence result of Sublinear Herding, a formal version of Theorem 3.2)**.** *For any parameters ϵ, δ, there is an iterative algorithm (Algorithm 4) that takes $O(dn^{1+o(1)} \cdot \kappa)$ time in pre-processing and $O((n^{1+o(1)} + dn) \cdot \kappa)$ space, takes $T = O(\frac{D_{\max}^2}{c^2 \epsilon})$ iterations and $O(dn^{\rho} \cdot \kappa)$ cost per iteration, starts from a random w^0 from $\mathcal{B}$ as initialization point, updates the w in each iteration based on Algorithm 4 and outputs $w^T \in \mathbb{R}^d$ from $\mathcal{B}$ such that*

$$\frac{1}{2}\|w^T - \mu\|_2^2 - \min_{w \in \mathcal{B}} \frac{1}{2}\|w - \mu\|_2^2 \leq \epsilon,$$

holds with probability at least $1 - \delta$. Here $\rho := \frac{2(1-\tau)^2}{(1-c\tau)^2} - \frac{(1-\tau)^4}{(1-c\tau)^4} + o(1)$ and $\kappa := \Theta(\log(T/\delta))$.

Proof. First, we show that $g(w) = \frac{1}{2}\|w^T - \mu\|_2^2$ is a convex and 1-smooth function. using Lemma E.2. Then, we could prove the theorem using Theorem E.3. Following the fact that the computation of gradient is $O(d)$, we could also provide the query time, preprocesisng time and space complexities.

□

E.3 Discussion

We show that our sublinear Frank-Wolfe algorithm demonstrated in Algorithm 4 breaks the linear cost per iteration of current Frank-Wolfe algorithm in Algorithm 3 in herding algorithm. Meanwhile, the extra number of iterations Algorithm 4 pay is affordable.

F Policy Gradient Optimization

We present the our results on policy gradient in this section.

F.1 Problem Formulation

In this paper, we focus on the action-constrained Markov Decision Process (ACMDP). In this setting, we are provided with a state $\mathcal{S} \in \mathbb{R}^k$ and action space $\mathcal{A} \in \mathbb{R}^d$. However, at each step $t \in \mathbb{N}$, we could only access a finite subset of actions $\mathcal{C}(s) \subset \mathcal{A}$ with cardinality n. Let us denote $D_{\max}$ as the maximum diameter of $\mathcal{A}$.

When you play with this ACMDP, the policy you choose is defined as $\pi_\theta(s) : \mathcal{S} \to \mathcal{A}$ with parameter θ. Meanwhile, there exists a reward function $r : \mathcal{S} \times \mathcal{A} \in [0, 1]$. Next, we define the Q function as below,

$$Q(s, a|\pi_\theta) = \mathbb{E}\Big[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t) | s_0 = s, a_0 = a, \pi_\theta\Big].$$

where $\gamma \in (0, 1)$ is a discount factor.

Given a state distribution μ, the objective of policy gradient is to maximize the expected value $J(\mu, \pi_\theta) = \mathbb{E}_{s \sim \mu, a \sim \pi_\theta}[Q(s, a | \pi_\theta)]$ via policy gradient [58] denoted as:

$$\nabla_\theta J(\mu, \pi_\theta) = \mathop{\mathbb{E}}_{s \sim d_\mu^\pi} \Big[\nabla_\theta \pi_\theta(s) \nabla_a Q(s, \pi_\theta(s) | \pi_\theta) | \Big].$$

[5] propose an iterative algorithm that perform MaxIP at each iteration k over actions to find

$$g_k(s) = \max_{a \in \mathcal{C}(s)} \langle a_s^k - \pi_\theta^k(s), \nabla_a Q(s, \pi_\theta^k(s) | \pi_\theta^k)) \rangle. \tag{26}$$

Moreover, [5] also have the following statement

Lemma F.1 ([5])**.** *Given a ACMDP and the gap $g_k(s)$ in Eq.(26), we show that*

$$J(\mu, \pi_\theta^{k+1}) \geq J(\mu, \pi_\theta^k(s)) + \frac{(1-\gamma)^2 \mu_{\min}^2}{2LD_{\max}^2} \sum_{s \in \mathcal{S}} g_k(s)^2$$

Therefore, [5] maximize the expected value via minimizing $g_k(s)$.

In this work, we accelerate Eq. (6) using (c, ϕ, ψ, τ)-MaxIP. Here define $\phi : \mathcal{S} \times \mathbb{R}^d \to \mathbb{R}^{d+2}$ and $\psi : \mathbb{R}^d \to \mathbb{R}^{d+3}$ as follows:

Corollary F.2 (Transformation for Policy Gradient)**.** *Let g be a differential function defined on convex set $\mathcal{K} \subset \mathbb{R}^d$ with maximum diameter $D_\mathcal{K}$. For any $x, y \in \mathcal{K}$, we define $\phi, \psi : \mathbb{R}^d \to \mathbb{R}^{d+3}$ as follows:*

$$\phi(x) := \begin{bmatrix} \frac{\phi_0(x)^\top}{D_x} & 0 & \sqrt{1 - \frac{\|\phi_0(x)\|_2^2}{D_x^2}} \end{bmatrix}^\top \quad \psi(y) := \begin{bmatrix} \frac{\psi_0(y)^\top}{D_y} & \sqrt{1 - \frac{\|\psi_0(y)\|_2^2}{D_y^2}} & 0 \end{bmatrix}^\top$$

where

$$\begin{aligned} \phi_0(s, \pi_\theta^k) &:= [\nabla_a Q(s, \pi_\theta^k(s) | \pi_\theta^k)^\top, (\pi_\theta^k)^\top Q(s, \pi_\theta^k(s) | \pi_\theta^k)]^\top \\ \psi_0(a) &= [a^\top, -1]^\top \end{aligned}$$

and D_x is the maximum diameter of $\phi_0(x)$ and D_y is the maximum diameter of $\psi_0(y)$.

Then, for all $x, y \in \mathcal{K}$ we have $g_k(s) = D_x D_y \langle \phi(s, \pi_\theta^k), \psi(a) \rangle$. Moreover, $\phi(x)$ and $\psi(y)$ are unit vectors with norm 1.

Proof. We show that

$$\begin{aligned} \langle \phi(s, \pi_\theta^k), \psi(a) \rangle &= D_x^{-1} D_y^{-1} \langle \nabla_a Q(s, \pi_\theta^k(s) | \pi_\theta^k), a \rangle - \langle \nabla_a Q(s, \pi_\theta^k(s) | \pi_\theta^k), \pi_\theta^k \rangle \\ &= D_x^{-1} D_y^{-1} \langle a_s^k - \pi_\theta^k(s), \nabla_a Q(s, \pi_\theta^k(s) | \pi_\theta^k)) \rangle \end{aligned}$$

where the first step follows the definition of ϕ and ψ, the second step is an reorganization. □

In this way, we propose a sublinear iteration cost algorithm for policy gradient in Algorithm 5.

Algorithm 5 Sublinear Frank-Wolfe Policy Optimization (SFWPO)

```
1: data structure LSH                                                    ▷ Corollary B.2
2:     INIT($S \subset \mathbb{R}^d$, $n \in \mathbb{N}$, $d \in \mathbb{N}$, $c \in (0,1)$)
3:         ▷ $|S| = n$, $c \in (0,1)$ is LSH parameter, and $d$ is the dimension of data
4:     QUERY($x \in \mathbb{R}^d$, $\tau \in (0,1)$)                     ▷ $\tau \in (0,1)$ is LSH parameter
5: end data structure
6:
7: procedure SFWPO($\mathcal{S} \subset \mathbb{R}^k$, $c \in (0,1)$,$\tau \in (0,1)$)
8:                                                                        ▷ Theorem F.3
9:     Input: Initialize the policy parameters as $\theta_0 \in R^l$ that satisfies $\pi_\theta^0(s) \in \mathcal{C}(s)$ for all $s \in \mathcal{S}$
10:    for each State $s \in \mathcal{S}$ do
11:        Construct $\phi, \psi : \mathbb{R}^d \to \mathbb{R}^{d+1}$ as Corollary F.2
12:        static LSH $\mathrm{LSH}_s$
13:        $\mathrm{LSH}_s$ INIT($\psi(\mathcal{C}(s), n, d+3, c)$
14:    end for
15:    $T \leftarrow O(\frac{c^{-2} L D_{\max}^2}{\epsilon^2 (1-\gamma)^3 \mu_{\min}^2}$
16:    for each iteration $k = 0, 1, \cdots, T$ do
17:        for each State $s \in \mathcal{S}$ do
18:            Use policy $\pi_\theta^k$ and obtain $Q(s, \pi_\theta^k(s)|\pi_\theta^k)$
19:        end for
20:        for each State $s \in \mathcal{S}$ do
21:            $\widehat{a_s^k} \leftarrow \mathrm{LSH}_s$.QUERY($\phi(s, \pi_\theta^k(s), \tau)$)
22:            $\widehat{g_k}(s) = \langle \widehat{a_s^k} - \pi_\theta^k(s), \nabla_a Q(s, \pi_\theta^k(s)|\pi_\theta^k)) \rangle$
23:            $\alpha_k(s) = \frac{(1-\gamma)\mu_{\min}}{L D_s^2} \widehat{g_k}(s)$
24:            $\pi_\theta^{k+1}(s) = \pi_\theta^k(s) + \alpha_k(s)(\widehat{a_s^k} - \pi_\theta^k(s))$
25:        end for
26:    end for
27:    return $\pi_\theta^T(s)$
28: end procedure
```

F.2 Convergence Analysis

The goal of this section is to show the convergence analysis of of Algorithm 5 compare it with [5].We first show the comparison results in Table 4.

Algorithm	Statement	Preprocessing	#iters	cost per iter
[5]	[5]	0	$O(\frac{\beta D_{\max}^2}{\epsilon^2 (1-\gamma)^3 \mu_{\min}^2})$	$O(dn + \mathcal{T}_Q)$
Algorithm 5	Theorem F.3	$O(dn^{1+o(1)} \cdot \kappa)$	$O(\frac{c^{-2} \beta D_{\max}^2}{\epsilon^2 (1-\gamma)^3 \mu_{\min}^2})$	$O(dn^\rho \cdot \kappa + \mathcal{T}_Q)$

Table 4: Comparison between our sublinear policy gradient (Algorithm 5) and [5].

The goal of this section is to prove Theorem F.3.

Theorem F.3 (Sublinear Frank-Wolfe Policy Optimization (SFWPO), a formal version of Theorem 3.3)**.** *Let $\mathcal{T}_Q$ denotes the time for computing the policy graident. Let $D_{\max}$ denotes the maximum diameter of action space and β is a constant. Let $\gamma \in (0,1)$. Let $\rho \in (0,1)$ denotes a fixed parameter. Let $\mu_{\min}$ denotes the minimal density of sates in $\mathcal{S}$. There is an iterative algorithm (Algorithm 5) that spends $O(dn^{1+o(1)} \cdot \kappa)$ time in preprocessing and $O((n^{1+o(1)} + dn) \cdot \kappa)$ space, takes $O(\frac{\beta D_{\max}^2}{\epsilon^2 (1-\gamma)^3 \mu_{\min}^2})$ iterations and $O(dn^\rho \cdot \kappa + \mathcal{T}_Q)$ cost per iterations, start from a random point π_θ^0 as initial point, and output policy π_θ^T that have average gap $\sqrt{\sum_{s \in \mathcal{S}} g_T(s)^2} < \epsilon$ holds with probability at least $1 - 1/\operatorname{poly}(n)$, where $g_T(s)^2$ is defined in Eq. (26) and $\kappa := \Theta(\log(T/\delta))$.*

Proof. Let $\widehat{a_s^k}$ denotes the action retrieved by LSH. Note that similar to Case 1 of Theorem D.1, the algorithms convergences if parameter τ is greater than maximum inner product. Therefore, we could direct focus on Case 2 and lower bound $\widehat{g_k}(s)$ as

$$\begin{aligned} \widehat{g_k}(s) &= \langle \widehat{a_s^k} - \pi_\theta^k(s), \nabla_a Q(s, \pi_\theta^k(s)|\pi_\theta^k)) \rangle \\ &= D_x D_y \langle \phi(s, \pi_\theta^k), \psi(\widehat{a_s^k}) \rangle \\ &\geq c D_x D_y \max_{a \in \mathcal{C}(a)} \langle \phi(s, \pi_\theta^k), \psi(a) \rangle \\ &= c\langle a_s^k, \nabla_a Q(s, \pi_\theta^k(s)|\pi_\theta^k)) \rangle - c\langle \pi_\theta^k(s), \nabla_a Q(s, \pi_\theta^k(s)|\pi_\theta^k)) \rangle \\ &= c g_k(s) \end{aligned} \tag{27}$$

where the first step follows from the line 22 in Algorithm 5, the second step follows from Corollary F.2, the third step follows from Corollary B.2, the forth step follows from Corollary F.2 and the last step is a reorganization.

Next, we upper bound $J(\mu, \pi_\theta^{k+1})$ as

$$\begin{aligned} J(\mu, \pi_\theta^{k+1}) &\geq J(\mu, \pi_\theta^k(s)) + \frac{(1-\gamma)^2 \mu_{\min}^2}{2LD_{\max}^2} \sum_{s \in \mathcal{S}} \widehat{g_k}(s)^2 \\ &\geq J(\mu, \pi_\theta^k(s)) + \frac{c^2(1-\gamma)^2 \mu_{\min}^2}{2LD_{\max}^2} \sum_{s \in \mathcal{S}} g_k(s)^2 \end{aligned} \tag{28}$$

where the first step follows from Lemma F.1, the second step follows from Eq. (27)

Using induction from 1 to T, we have

$$J(\mu, \pi_\theta^T) = J(\mu, \pi_\theta^1) + \frac{c^2(1-\gamma)^2 \mu_{\min}^2}{2LD_{\max}^2} \sum_{k=0}^{T} \sum_{s \in \mathcal{S}} g_k(s)^2 \tag{29}$$

Let $G = \sum_{k=0}^T \sum_{s \in \mathcal{S}} g_k(s)^2$, we upper bound G as

$$\begin{aligned} G &\leq \frac{2LD_{\max}^2}{c^2(1-\gamma)^2 \mu_{\min}^2} (J(\mu, \pi_\theta^T) - J(\mu, \pi_\theta^0)) \\ &\leq \frac{2LD_{\max}^2}{c^2(1-\gamma)^2 \mu_{\min}^2} J(\mu, \pi_\theta^*)) \\ &\leq \frac{2LD_{\max}^2}{c^2(1-\gamma)^3 \mu_{\min}^2} \end{aligned} \tag{30}$$

where the first step follows from Eq (29), the second step follows from $J(\mu, \pi_\theta^*) \geq J(\mu, \pi_\theta^T)$, last step follows from $J(\mu, \pi_\theta^*) \leq (1-\gamma)^{-1}$.

Therefore, we upper bound $\sum_{s \in \mathcal{S}} g_T(s)^2$ as

$$\begin{aligned} \sum_{s \in \mathcal{S}} g_T(s)^2 &\leq \frac{1}{T+1} G \\ &\leq \frac{1}{T+1} \frac{2LD_{\max}^2}{c^2(1-\gamma)^3 \mu_{\min}^2} \end{aligned} \tag{31}$$

where the first step is a reorganization, the second step follows that $\sum_{s \in \mathcal{S}} g_T(s)^2$ is non-increasing, the second step follows from Eq (30).

If we want $\sum_{s \in \mathcal{S}} g_T(s)^2 < \epsilon^2$, T should be $O(\frac{c^{-2} L D_{\max}^2}{\epsilon^2 (1-\gamma)^3 \mu_{\min}^2})$

Preprocessing time According to Corrollary B.2, can construct $\kappa = \Theta(\log(T/\delta))$ LSH data structures for (c, ϕ, ψ, τ)-MaxIP with ϕ, ψ defined in Corollary F.2. As transforming every $a \in \mathcal{A}$ into $\psi(a)$ takes $O(dn)$. Therefore, the total the preprocessing time complexity is $O(dn^{1+o(1)} \cdot \kappa)$.

Cost per iteration

Given each w^t, compute the policy gradient takes $\mathcal{T}_Q$. Next, it takes $O(d)$ time to generate $\phi(s, \pi_\theta^k)$ according to Corollary C.3 based on policy gradient. Next, according to Corrollary B.2, it takes $O(dn^\rho \cdot \kappa)$ to retrieve action from $\kappa = \Theta(\log(T/\delta))$ LSH data structures. After we select action, it takes $O(d)$ time to compute the gap the update the value. Thus, the total complexity is $O(dn^\rho \cdot \kappa + \mathcal{T}_Q)$ with $\rho := \frac{2(1-\tau)^2}{(1-c\tau)^2} - \frac{(1-\tau)^4}{(1-c\tau)^4} + o(1)$.

□

F.3 Discussion

We show that our sublinear Frank-Wolfe based policy gradient algorithm demonstrated in Algorithm 5 breaks the linear cost per iteration of current Frank-Wolfe based policy gradient algorithm algorithm. Meanwhile, the extra number of iterations Algorithm 5 pay is affordable.

G More Data Structures: Adaptive MaxIP Queries

In optimization, the gradient at each iteration is not independent from the previous gradient. Therefore, it becomes a new setting for using (c, τ)-MaxIP. If we take the gradient as query and the feasible set as the data set, the queries in each step forms an adaptive sequence. In this way, the failure probability of LSH or other (c, τ)-MaxIP data-structures could not be union bounded. To extend (c, τ)-MaxIP data-structures such as LSH and graphs into this new setting, we demonstrate a query quantization method.

We start with relaxing the (c, τ)-MaxIP with a inner product error.

Definition G.1 (Relaxed approximate MaxIP)**.** *Let approximate factor $c \in (0,1)$ and threshold $\tau \in (0,1)$. Let $\lambda \geq 0$ denotes an additive error. Given an n-vector set $Y \subset \mathbb{S}^{d-1}$, the objective of (c, τ, λ)-MaxIP is to construct a data-structure that, for a query $x \in \mathbb{S}^{d-1}$ with conditions that $\max_{y \in Y} \langle x, y \rangle \geq \tau$, it retrieves vector $z \in Y$ that $\langle x, z \rangle \geq c \cdot \mathsf{MaxIP}(x, Y) - \lambda$.*

Then, we present a query quantization approach to solve (c, τ, λ)-MaxIP for adaptive queries. We assume that the Q is the convex hull of all queries. For any query $x \in Q$, we perform a quantization on it and locate it to the nearest lattice with center $\widehat{q} \in Q$. Here the lattice has maximum diameter 2λ. Then, we query $\widehat{q}$ on data-structures e.g., LSH, graphs, alias tables. This would generate a λ additive error to the inner product. Because the lattice centers are independent, the cumulative failure probability for adaptive query sequence could be union bounded. Formally, we present the corollary as

Corollary G.2 (A query quantization version of Corollary B.1)**.** *Let approximate factor $c \in (0,1)$ and threshold $\tau \in (0,1)$. Given a n-vector set $Y \subset \mathbb{S}^{d-1}$, one can construct a data-structure with $O(dn^{1+o(1)} \cdot \kappa)$ preprocessing time and $O((n^{1+o(1)} + dn) \cdot \kappa)$ space so that for every query x in an adaptive sequence $X = \{x_1, x_2, \cdots, x_T\} \subset \mathbb{S}^{d-1}$, we take query time complexity $O(dn^\rho \cdot \kappa)$ to solve (c, τ, λ)-MaxIP with respect to (x, Y) with probability at least $1 - \delta$, where $\rho = \frac{2(1-\tau)^2}{(1-c\tau)^2} - \frac{(1-\tau)^4}{(1-c\tau)^4} + o(1)$, $\kappa := d \log(ndD_X/(\lambda\delta))$ and D_X is the maximum diameter in ℓ_2 distance of all queries in X.*

Proof. The probability that at least one query $x \in X$ fails is equivalent to the probability that at least one query $\widehat{q} \in \widehat{Q}$ fails. Therefore, we could union bound the probability as:

$$\Pr[\exists \widehat{q} \in \widehat{Q} \quad \text{s.t all } \ (c, \tau)\text{-}\mathsf{MaxIP} \text{ fail}] = n \cdot (\frac{dD_X}{\lambda})^d \cdot (1/10)^\kappa \leq \delta$$

where the second step follows from $\kappa := d \log(ndD_X/(\lambda\delta))$.

The results of $\widehat{q}$ has a λ additive error to the original query. Thus, our results is a (c, τ, λ)-MaxIP solution. The time and space complexty is obtained via Corollary B.1. Thus we finish the proof. □

Definition G.3 (Quantized projected approximate MaxIP)**.** *Let approximate factor $c \in (0,1)$ and threshold $\tau \in (0,1)$. Let $\lambda \geq 0$ denotes an additive error. Let $\phi, \psi : \mathbb{R}^d \rightarrow \mathbb{R}^k$ denotes two* *transforms. Given an n-point dataset $Y \subset \mathbb{R}^d$ so that $\psi(Y) \subset \mathbb{S}^{d-1}$, the goal of the $(c, \phi, \psi, \tau, \lambda)$-*MaxIP *is to build a data structure that, given a query $x \in \mathbb{R}^d$ and $\phi(x) \in \mathbb{S}^{k-1}$ with the promise that $\max_{y \in Y}\langle \phi(x), \psi(y)\rangle \geq \tau - \lambda$, it retrieves a vector $z \in Y$ with $\langle \phi(x), \psi(z)\rangle \geq c \cdot (\phi, \psi)$-*MaxIP$(x, Y)$.

Next, we extend Corollary G.2 to adaptive queries.

Corollary G.4. *Let $c \in (0, 1)$, $\tau \in (0, 1)$, $\lambda \geq 0$ and $\delta \geq 0$. Let $\phi, \psi : \mathbb{R}^d \to \mathbb{R}^k$ denotes two transforms. Let $\mathcal{T}_\phi$ denotes the time to compute $\phi(x)$ and $\mathcal{T}_\psi$ denotes the time to compute $\psi(y)$. Given a set of n-points $Y \in \mathbb{R}^d$ with $\psi(Y) \subset \mathcal{S}^{k-1}$ on the sphere, one can construct a data structure with $O(dn^{1+o(1)} \cdot \kappa + \mathcal{T}_\psi n)$ preprocessing time and $O((dn^{1+o(1)} + dn) \cdot \kappa)$ space so that for any query $x \in \mathbb{R}^d$ with $\phi(x) \in \mathcal{S}^{k-1}$, we take query time complexity $O(dn^\rho \cdot \kappa + \mathcal{T}_\phi)$ to solve $(c, \phi, \psi, \tau, \lambda)$-*MaxIP *with respect to (x, Y) with probability at least $1 - \delta$, where $\rho := \frac{2(1-\tau)^2}{(1-c\tau)^2} - \frac{(1-\tau)^4}{(1-c\tau)^4} + o(1)$, $\kappa := d \log(ndD_X/(\lambda\delta))$ and D_X is the maximum diameter in ℓ_2 distance of all queries in X.*

Finally, we present a modified version of Theorem D.1.

Theorem G.5 (Convergence result of Frank-Wolfe via LSH with Adaptive Input)**.** *Let $g : \mathbb{R}^d \to \mathbb{R}$ denotes a convex (see Definition A.10) and β-smooth function (see Definition A.9). Let the complexity of calculating $\nabla g(x)$ to be $\mathcal{T}_g$. Let $S \subset \mathbb{R}^d$ denotes a set of points with $|S| = n$, and $\mathcal{B} \subset \mathbb{R}^d$ is the convex hull of S defined in Definition A.8. For any parameters ϵ, δ, there is an iterative algorithm with $(c, \phi, \psi, \tau, c^{-2}\epsilon/4)$-*MaxIP *data structure that takes $O(dn^{1+o(1)} \cdot \kappa)$ preprocessing time and $O((n^{1+o(1)} + dn) \cdot \kappa)$ space, takes $T = O(\frac{\beta D_{\max}^2}{\epsilon})$ iterations and $O(dn^\rho \cdot \kappa + \mathcal{T}_g)$ cost per iteration, starts from a random w^0 from $\mathcal{B}$ as initialization point, updates the w in each iteration as follows:*

$$
\begin{aligned}
s^t &\leftarrow (c, \phi, \psi, \tau, c^{-2}\epsilon/4)\text{-MaxIP } \textit{of } w^t \textit{ with respect to } S \\
w^{t+1} &\leftarrow w^t + \eta \cdot (s^t - w^t)
\end{aligned}
$$

and outputs $w^T \in \mathbb{R}^d$ from $\mathcal{B}$ such that

$$g(w^T) - \min_{w \in \mathcal{B}} g(w) \leq \epsilon,$$

holds with probability at least $1 - \delta$. Here $\kappa := d \log(ndD_X/(\lambda\delta))$ and $\rho := \frac{2(1-\tau)^2}{(1-c\tau)^2} - \frac{(1-\tau)^4}{(1-c\tau)^4} + o(1)$.

Proof. **Convergence.**

We start with modifying Eq. (14) with additive MaxIP error λ and get

$$h_{t+1} = (1 - c\eta_t)h_t + \frac{\beta D_{\max}^2}{2}\eta_t^2 + \eta_t\lambda$$

Let $e_t = A_t h_t$ with $A_t = \frac{t(t+1)}{2}$. Let $\eta_t = \frac{2}{c(t+2)}$. Let $\lambda = \frac{\beta D_{\max}^2}{T+1}$ Following the proof in Theorem D.1, we upper bound $e_{t+1} - e_t$ as

$$e_{t+1} - e_t \leq (A_{t+1}(1 - c\eta_t) - A_t)\, h_t + \frac{\beta D_{\max}^2}{2} A_{t+1}\eta_t^2 + A_{t+1}\eta_t\lambda \tag{32}$$

where

- $A_{t+1}(1 - \eta_t) - A_t = 0$
- $A_{t+1}\frac{\eta_t^2}{2} = \frac{t+1}{(t+2)c^2} < c^{-2}$
- $A_{t+1}\eta_t\lambda = (t+1)\lambda < \beta D_{\max}^2$.

Therefore,

$$e_{t+1} - e_t < 2c^{-2}\beta D_{\max}^2 \tag{33}$$

Based on Eq (33), we upper bound e_t using induction and have

$$e_t < 2c^{-2}t\beta D_{\max}^2 \tag{34}$$

Using the definition of e_t, we have

$$h_t = \frac{e_t}{A_t} < \frac{4\beta D_{\max}^2}{c^2(t+1)} \tag{35}$$

To make $h_T \le \epsilon$, T should be in $O(\frac{\beta D_{\max}^2}{c^2\epsilon})$. Moreover, $\lambda = \frac{\beta D_{\max}^2}{T+1} = \frac{\epsilon}{4c^2}$.

Preprocessing time According to Corrollary G.4, can construct $\kappa = d\log(ndD_X/(\lambda\delta))$ LSH data structures for $(c, \phi, \psi, \tau, c^{-2}\epsilon/4)$-MaxIP with ϕ, ψ defined in Corollary C.3. As transforming every $s \in S$ into $\psi(s)$ takes $O(dn)$. Therefore, the total the preprocessing time complexity is $O(dn^{1+o(1)} \cdot \kappa)$ and space complexity is $O((n^{1+o(1)} + dn) \cdot \kappa)$.

Cost per iteration

Given each w^t, compute $\nabla g(w^t)$ takes $\mathcal{T}_g$. Next, it takes $O(d)$ time to generate $\phi(w^t)$ according to Corollary C.3 based on $g(w^t)$ and $\nabla g(w^t)$. Next, according to Corrollary G.4, it takes $O(dn^\rho \cdot \kappa)$ to retrieve s^t from κ LSH data structures. After we select s^t, it takes $O(d)$ time to update the w^{t+1}. Combining the time for gradient calculation, LSH query and w^t update, the total complexity is $O(dn^\rho \cdot \kappa + \mathcal{T}_g)$ with $\rho := \frac{2(1-\tau)^2}{(1-c\tau)^2} - \frac{(1-\tau)^4}{(1-c\tau)^4} + o(1)$.

□

Similarly, we could extend the results to statements of Herding algorithm and policy gradient.

Theorem G.6 (Modified result of Sublinear Herding,)**.** *For any parameters ϵ, δ, there is an iterative algorithm (Algorithm 4) with $c^{-2}\epsilon/4$ query quantization that takes $O(dn^{1+o(1)} \cdot \kappa)$ time in preprocessing and $O((n^{1+o(1)} + dn) \cdot \kappa)$ space, takes $T = O(\frac{D_{\max}^2}{c^2\epsilon})$ iterations and $O(dn^\rho \cdot \kappa)$ cost per iteration, starts from a random w^0 from $\mathcal{B}$ as initialization point, updates the w in each iteration based on Algorithm 4 and outputs $w^T \in \mathbb{R}^d$ from $\mathcal{B}$ such that*

$$\frac{1}{2}\|w^T - \mu\|_2^2 - \min_{w\in\mathcal{B}} \frac{1}{2}\|w - \mu\|_2^2 \le \epsilon,$$

holds with probability at least $1 - \delta$. Here $\rho := \frac{2(1-\tau)^2}{(1-c\tau)^2} - \frac{(1-\tau)^4}{(1-c\tau)^4} + o(1)$ and $\kappa := d\log(ndD_X/(\lambda\delta))$.

Theorem G.7 (Modified result of Sublinear Frank-Wolfe Policy Optimization (SFWPO))**.** *Let $\mathcal{T}_Q$ denotes the time for computing the policy graident. Let $D_{\max}$ denotes the maximum diameter of action space and β is a constant. Let $\gamma \in (0,1)$. Let $\rho \in (0,1)$ denotes a fixed parameter. Let $\mu_{\min}$ denotes the minimal density of sates in $\mathcal{S}$. There is an iterative algorithm (Algorithm 5) with $c^{-2}\epsilon/4$ query quantization that spends $O(dn^{1+o(1)} \cdot \kappa)$ time in preprocessing and $O((n^{1+o(1)} + dn) \cdot \kappa)$ space, takes $O(\frac{\beta D_{\max}^2}{\epsilon^2(1-\gamma)^3\mu_{\min}^2})$ iterations and $O(dn^\rho \cdot \kappa + \mathcal{T}_Q)$ cost per iterations, start from a random point π_θ^0 as initial point, and output policy π_θ^T that have average gap $\sqrt{\sum_{s\in\mathcal{S}} g_T(s)^2} < \epsilon$ holds with probability at least $1 - 1/\operatorname{poly}(n)$, where $g_T(s)$ is defined in Eq. (26) and $\kappa := d\log(ndD_X/(\lambda\delta))$.*