Faster Algorithm for One-Sided Kadison-Singer via Furthest-Neighbor Search

Zhao Song[*] Zhaozhuo Xu[†] Lichen Zhang[‡]

Abstract

We study the algorithmic version of Kadison-Singer problem under relaxed one-sided guarantee proposed in [Wea13]. The major contribution of this paper is a novel framework that combines iterative analysis for discrepancy problem with one-sided guarantee and approximate furthest-neighbor search data structure. Intuitively, combining iterative scheme and nearest-neighbor search could give rise to fast greedy algorithms. However, we show that to solve one-sided discrepancy-type problem, we need the furthest-neighbor search data structure. Our framework can be extended to more one-sided discrepancy problems such as the rounding up task in experimental design problem. Another interesting application is finding matrix rows with small leverage scores, which is a key sub-task in cutting plane method [JLSW20].

To the best of our knowledge, this is the first work that explicitly studies one-sided Kadison-Singer problem from an algorithmic perspective. Moreover, our algorithm improves upon existing approaches through the novel co-design of data structure and optimization.

[*]`zhaos@ias.edu`. Institute for Advanced Study.

[†]`zx22@rice.edu`. Rice University.

[‡]`lichenz@andrew.cmu.edu`. Carnegie Mellon University.

1 Introduction

In mathematics, the famous Kadison-Singer problem [KS59] was a functional analysis problem posed in 1959. Formally speaking, the problem is as follows:

Question 1.1 (Kadison-Singer problem)**.** *Does every pure state on the (abelian) von Neumann algebra $\mathcal{D}$ of bounded diagonal opeartors on ℓ_2 have a unique extension to a pure state on $B(\ell_2)$, the von Neumann algebra of all bounded operators on ℓ_2?*

Various conjectures and statements are shown to be equivalent to the above question, for example, Anderson's paving conjectures [And79, And81], Weaver's discrepancy theoretical formulation [Wea04]. For a discussion of related conjectures, we refer readers to [CT06].

In 2014, a celebrated result by Marcus, Spielman and Srivastava [MSS15] has given a positive answer to the Kadison-Singer problem. Specifically, they provided a affirmative answer to Weaver's discrepancy conjecture, which is now a theorem:

Theorem 1.2 ([Wea04, MSS15])**.** *There exists universal constants $s \geq 2$ and $\theta \geq 0$, so that we have the following: given a set of vectors $v_1, \ldots, v_m \in \mathbb{C}^d$ satisfying $\|v_i\| \leq 1$ for all i, and suppose that $\sum_{i=1}^m |\langle u, v_i\rangle|^2 = s$. Then there exists a partition S_1, S_2 of $\{1, \ldots, m\}$ such that for any unit vector $u \in \mathbb{C}^d$,*

$$\sum_{i \in S_j} |\langle u, v_i\rangle|^2 \leq s - \theta, \quad j = 1, 2.$$

Their main result, which implies a positive answer to Theorem 1.2, can be viewed as a matrix concentration result similar to [Rud96, AW02] for rank 1 matrices, but much stronger. Their discrepancy result is further strengthened or generalized in [Coh16, Brä18, KLS20].

In this paper, we try to solve Kadison-Singer problem from an algorithmic perspective. Specifically, we observe that Theorem 1.2 can be formulated as an algorithmic problem: Suppose we are given a set of m vectors $v_1, \ldots, v_m \in \mathbb{C}^d$ satisfying the conditions of Weaver's discrepancy formulation, then, how fast can we find the desired partition S_1, S_2 of $\{1, \ldots, m\}$? We note that the proof in [MSS15] can be easily extended to an exponential time algorithm by examining all subsets of $\{1, \ldots, m\}$, which takes $O(2^m)$ time. By improving the running time of approximating the largest root of a characteristic polynomial, [AGSS18] gives an $2^{\widetilde{O}(m^{1/3})}$ time algorithm. However, it still remains open whether it is possible to solve the algorithmic problem posed in polynomial or even quasi-polynomial time. Instead of trying to obtain an exponential speedup, we take a step back and consider a simpler version of the algorithmic Kadison-Singer problem, which is introduced in [Wea13].

Note that in the original formulation proposed by Weaver, the resulting partition can be viewed to have a two-sided guarantee, namely, we are looking for a subset S such that

$$\theta \leq \sum_{i \in S} |\langle u, v_i\rangle|^2 \leq s - \theta. \tag{1}$$

This would naturally imply a similar upper bound guarantee on $[m] \setminus S$. A simpler task, which is called one-sided version of Kadison-Singer, is to find a subset S that only satisfies the upper bound of Eq. (1). This means we can only get a lower bound on $[m]\setminus S$, the upper bound can be much larger than $s - \theta$. Such a simplification does admit a polynomial time algorithm, which can be derived from the proof of Theorem 3.3 in [Wea13]. The algorithm constructs set S in a greedy fashion, it searches for a vector that is "small" based on certain measure and adds it to S. Interestingly, there are plenty of discrepancy type problems that seek one-sided guarantee in this format, such as experimental design through regret minimization [AZLSW20]. Proposed algorithms for this kind of problems share similar greedy structure: at each iteration, computes certain measure against each vector and find the one with smallest/largest measure. Even for small d regime, it seems inevitable that we have to pay at least $\Omega(m)$ cost per iteration. We ask the following question:

Does there exist a class of algorithms that can break the $O(m)$ barrier per iteration?

To break this barrier, we follow a growing trend in the field of optimization algorithm: using efficient data structures to speedup expensive operations in an iterative regime, as popularized in [CLS19, LSZ19, JLSW20, Bra20, BLSS20, DLY21, SY21, JSWZ21, Bra21]. We point out that our technique is completely novel, in the sense that 1). it reduces an optimization objective into a general search problem, 2). solving the search problem through the use of efficient data structure.

In this work, we focus on a class of data structures which is less popular and prevalent, namely, the *furthest-neighbor search* (FN), or *minimum inner product* (Min-IP) data structures. To start with, notice one can formulate one-sided Kadison-Singer as the following discrepancy problem:

Given a set of points, finding a subset so that the sum of their inner products with points on unit circle are minimized, or in other words, the sum of their distances to unit circle is maximized.

Put it more plainly, one can think of it as a packing game: we are given a box with fixed volume and a bunch of balls with different volumes. The game is to pack as many balls into the box as possible. Furthest-neighbor search data structure will always find an exact or approximate ball with small inner product/volume, thus, we can pack a small ball into the box and give us more room for the remaining balls. We remark that such a simple game captures a wide range of interesting one-sided problems in discrepancy theory, for example, one-sided Kadison-Singer problem, experimental design problem through regret minimization [AZLSW20]. In all these tasks, one seeks a subset of vectors that gives a one-sided guarantee, typically in the form of

$$\left\| \sum_{i \in S} v_i v_i^\top \right\| \leq a$$

for some positive number a. As we will show in this paper, our scheme with furthest-neighbor search can be used to improve the running time of the deterministic procedure in [AZLSW20]. In their task, they are given a fractional solution $\pi \in [0,1]^m$, the goal is to round up to an integral solution with support at most $\|\pi\|_1$, such a task can be reformulated into the following: given $\sum_{i=1}^m \pi_i v_i v_i^\top = I_d$, finding a subset of cardinality at most $\|\pi\|_1$ such that

$$\lambda_{\min}\Big(\sum_{i \in S} v_i v_i^\top \Big) \geq 1 - 3\varepsilon.$$

In order to solve this task, they proposed a swapping algorithm that greedily constructs set S. At each round, they look for a vector in S with small inner product, and another vector in $[m] \setminus S$ with larger inner product, then swap them. In the regime where n is large, we can adapt our Min-IP data structure for the in-S search.

We will also show that such a scheme can be deployed in the field of optimization. As an example, consider cutting plane method (CPM) that is extensively studied and used for many problems where solving convex programming is a task or a necessary subtask [LW17, ADLS17, Jia21]. One of the key steps in CPM is to reconstructing the constraint matrix through separation oracle. As shown in [JLSW20], sometimes it is necessary to remove redundant constraints through leverage score. Surprisingly, our framework can be used to accelerate the step of computing small leverage score, as leverage score computation is essentially computing inner product, and furthest-neighbor search data structures are efficient under insertion and deletion.

2 Our Results

Throughout this section, we use high probability to denote $1-\frac{1}{\mathrm{poly}(m)}$ and we use O_d to hide $\mathrm{poly}(d)$ factors and $\widetilde{O}_d$ to hide $\mathrm{poly}(d, \log m)$ factors.

2.1 Main Results

We start by presenting our exact algorithm for algorithmic one-sided Kadison-Singer problem posed in [Wea13].

In the following result, we show that if pre-processing on vectors is allowed, we can improve the naive md time to $o(m)$ when dimension is d relatively small compared to m.

Theorem 2.1 (Exact algorithm, informal of Theorem 6.9)**.** *Let $N \in \mathbb{N}_+$, if $\{v_1, \dots, v_m\}$ is a finite sequence of vectors in $\mathbb{C}^d$ satisfying $\|v_i\|_2 = \frac{1}{\sqrt{N}}, \forall i \in [m]$ and $\sum_{i=1}^m |\langle u, v_i \rangle|^2 = 1$ holds for all unit vector $u \in \mathbb{C}^d$. Suppose we preprocess all the m vectors. Then for any $n < m$, there exists a deterministic algorithm that takes n iteration and each iteration takes $O_d(\log m)$ timeto find a set S with cardinality n such that*

$$\sum_{i \in S} |\langle u, v_i \rangle|^2 \leq \frac{n}{m} + O(\frac{1}{\sqrt{N}}).$$

The clear advantage of our deterministic algorithm is its query time only depends logarithmically on m. However, this kind of exact furthest-neighbor data structure makes use of Voronoi diagram, which suffers from curse of dimensionality, i.e., in initialization phase, it takes time exponential in dimension d, makes it only applicable when d is small. In contrary, our approximate algorithm has much milder dependence on d, makes it more applicable in a wider range of settings.

Theorem 2.2 (Approximate algorithm, informal of Theorem 6.11)**.** *Let $\tau, c \in (0,1)$ and $N \in \mathbb{N}_+$, if $\{v_1, \dots, v_m\}$ is a finite sequence of vectors in $\mathbb{C}^d$ satisfying $\|v_i\|_2 = \frac{1}{\sqrt{N}}, \forall i \in [m]$ and $\sum_{i=1}^m |\langle u, v_i \rangle|^2 = 1$ holds for all unit vector $u \in \mathbb{C}^d$. Then for any $n < m$ and unit vector $u \in \mathbb{C}^d$, there exists a randomized algorithm (success with high probability) that takes time $\mathcal{T}$ to find a set S ($|S| = n$) such that*

$$\sum_{i \in S} |\langle u, v_i \rangle|^2 \leq \frac{1}{c} \cdot (\frac{n}{m} + O(\frac{1}{\sqrt{N}})).$$

Further, we have

- *If $c \in (\tau, \frac{8\tau}{7+\tau})$, then $\mathcal{T} = \widetilde{O}_d(m^{1.5} + n \cdot m^{0.5})$;*
- *If $c \in (\tau, \frac{400\tau}{399+\tau})$, then $\mathcal{T} = \widetilde{O}_d(m^{1.01} + n \cdot m^{0.01})$.*

We make several remarks regarding the approximate algorithm theorem. First, note the two parameters τ and c serve as an approximate factor for the final solution, it is tempting to condense these two parameters into one. However, the parameter τ comes from the data structure, i.e., the data structure expects query of the type that finding the minimum inner product at most τ, and it will output an approximate solution with the guarantee that the inner product is at most τ/c. One expects a natural trade-off on c, i.e., a larger c implies a better quality of solution in the expense of a worse runtime. Such trade-off is also reflected in the theorem, i.e., if c can be chosen closer to 1 ($\frac{8\tau}{7+\tau} > \frac{400\tau}{399+\tau}$), then we get a better approximation with a worse running time.

The running time can be parsed into two parts: the $\widetilde{O}_d(m^{1.5})$ and $\widetilde{O}_d(m^{1.01})$ is the initialization time of furthest-neighbor/minimum inner product data structure, and the second term can be decomposed into the following:

- n is the number of iterations.
- d^ω is the time to compute certain matrix inverse at each iteration, which is absorbed by $\widetilde{O}_d(\cdot)$.
- $\widetilde{O}_d(m^{0.5})$ and $\widetilde{O}_d(m^{0.01})$ is the query time of data structure.

Note the initialization time no longer depends exponentially on d, makes it useful in most settings. The trade-off here is a slower query time ($m^{0.5}$ or $m^{0.01}$) and an approximate guarantee instead of exact solution. Below is a table comparing the runtime complexity of various algorithms:

Algorithm	**Prep.**	**Cost.**	**Total Time**	**Comments**
[Wea13]	0	m	$n \cdot m$	
Alg. 3	$m^{d^2/2}$	1	$n + m^{d^2/2}$	Our exact algorithm
Alg. 4	$m^{1.5}$	$m^{0.5}$	$n \cdot m^{0.5} + m^{1.5}$	Ours with high accuracy
Alg. 4	$m^{1.01}$	$m^{0.01}$	$n \cdot m^{0.01} + m^{1.01}$	Ours with low accuracy

Table 1: Comparison of different algorithms, for simplicity of presentation assume $m \gg d$ and ignore $\mathrm{poly}(d, \log m)$ factor. All algorithms require n iterations. **Prep.** denote the preprocessing time of data structures. **Cost.** denotes the cost spent in each iteration of the algorithm. Simplified version of Table 3.

For this problem, the first polynomial time algorithm is given in the proof of [Wea13], which is a simple greedy algorithm that takes time $O_d(nm)$. In contrast, our approximation scheme gives only depends on $nm^{0.5}$ or even $nm^{0.01}$, which is much more feasible if both m and n large. By utilizing an amortization argument, one can amortize the preprocessing/initialization time into n iterations, this means, e.g., in the high accuracy approximation scheme where it takes $\widetilde{O}_d(m^{1.5})$ to initialize, as long as $n \geq \widetilde{O}_d(m^{0.5})$, our algorithm is faster than algorithms not using such data structures. By trading runtime with accuracy, it is possible to achieve a even faster preprocessing and query time than $m^{1.01}$ and $m^{0.01}$. To the best of our knowledge, this is the fastest approximation algorithm to solve algorithmic one-sided Kadison-Singer problem.

Another result we have is to apply our $\mathsf{Min\text{-}IP}$ and AFN data structure to the experimental design problem through regret minimization as posed in [AZLSW20]. We provide a randomized and approximate algorithmic result for a more generalized version of this question:

Theorem 2.3 (Informal version of Theorem 7.16)**.** *Let $\pi \in [0,1]^m$ with $\|\pi\|_1 \leq n$ and $\sum_{i=1}^m \pi_i x_i x_i^\top = I_d$. Let $\gamma \geq 3$ and $\varepsilon \in (0, \frac{1}{\gamma}]$. Then, there exists a subset $S \subset [m]$ with $|S| \leq n$ such that*

$$\lambda_{\min}\Big(\sum_{i \in S} x_i x_i^\top\Big) \geq 1 - \gamma \cdot \varepsilon.$$

Let $\tau \in (0,1)$ and $c \in (\frac{1}{\gamma - 1}, 1)$. If $n \geq \frac{6d/\varepsilon^2}{\gamma - 1 - 1/c}$ and $\alpha = \sqrt{d}/(c\varepsilon)$, then there exists a randomized algorithm (success with high probability) that takes time $\mathcal{T}$ to find such S. Furthermore,[1]

- *If $c \in (\tau, \frac{8\tau}{7+\tau})$, then $\mathcal{T} = \widetilde{O}_d(m + \varepsilon^{-1} n^{1.5})$;*
- *If $c \in (\tau, \frac{400\tau}{399+\tau})$, then $\mathcal{T} = \widetilde{O}_d(m + \varepsilon^{-1} n^{1.01})$.*

Algorithm	**Prep.**	**Cost.**	**Total Time**
[AZLSW20]	m	m	$m + \varepsilon^{-1} nm$
Alg. 5	$m + n^{1.5}$	$n^{0.5}$	$m + \varepsilon^{-1} n^{1.5}$
Alg. 5	$m + n^{1.01}$	$n^{0.01}$	$m + \varepsilon^{-1} n^{1.01}$

Table 2: Comparison of different algorithms for experimental design, for simplicity we assume $n \gg m - n$ and ignore $\mathrm{poly}(d, \log m)$ factor. All algorithms require $\varepsilon^{-1} n$ iterations. **Prep.** denotes the preprocessing time. **Cost.** denotes the cost spent in each iteration of the algorithm. Simplified version of Table 4.

We make some observations of above theorem. In our approximate result for one-sided Kadison-Singer, where the approximate search gives a solution with worse guarantee. Here, we recover exactly the same guarantee as the vanilla greedy algorithm in the expense of a larger size of set S. This is an interesting but reasonable trade-off, since larger n implies we can add more potential vectors into the set, which makes the problem easier. In some sense, this is similar to have a solution with worse guarantee.

2.2 Technique Overviews

This section is dedicated to overview our algorithmic framework and the main techniques we employed. To start off, consider the upper barrier potential function studied in [Sri10, BSS12, Wea13, MSS15]:

Definition 2.4. Let T be a square matrix, for any $a > \|T\|$, we define the upper barrier potential function $\Phi^a(T)$ as $\Phi^a(T) := \mathrm{tr}[(aI - T)^{-1}]$.

Prior to its generalization to discrepancy problem, this class of potential functions has been widely utilized in finding a small size graph spectral sparsifier, see e.g., [BSS12, LS15, LS17].

The greedy algorithm of [Wea13] iteratively constructs the set S. Let S_t denote the set at around t, it also maintains a matrix $T := \sum_{i \in S_t} v_i v_i^\top$. At each round, it looks for an index $i^* \in [m] \setminus S_t$ such that

$$\frac{v_{i^*}^\top (a_{j+1} I - T)^{-2} v_{i^*}}{\Phi^{a_j}(T) - \Phi^{a_{j+1}}(T)} + v_{i^*}^\top (a_{j+1} I - T)^{-1} v_{i^*} \leq 1, \tag{2}$$

where a_j is a sequence of numbers defined inductively.

As we will show later, if we construct set S and maintain matrix T in a certain manner, we can guarantee there exists a vector satisfies Eq. (2), hence, we know the vector that gives the minimum value in the LHS of Eq. (2) must also satisfies the criteria.

Greedy Vector Selection as Minimum Inner Product Search. We observe that the LHS of Eq. (2) can be recast into the following inner product of two matrices:

$$\Big\langle \underbrace{v_{i^*} v_{i^*}^\top}_{\text{rank 1}}, \underbrace{\frac{(a_{j+1} I - T)^{-2}}{\Phi^{a_j}(T) - \Phi^{a_{j+1}}(T)} + (a_{j+1} I - T)^{-1}}_{\text{positive definite}} \Big\rangle, \tag{3}$$

[1]We consider the regime where $n \gg n - m$, e.g., $n = m - m^{o(1)}$ and $n - m = m^{o(1)}$, therefore we ignore the factor $(m - n) \cdot d^2$.

if we further vectorize these two matrices, it turns into a standard inner product between two vectors of dimension d^2. This means we can reduce the greedy construction of set S into a minimum inner product search (Min-IP) problem: design an efficient data structure that answers the query asking for a matrix/vector that has the minimum inner product with a bunch of preprocessed matrices.

Solving Minimum Inner Product Search via Furthest-Neighbor Search. We could solve Min-IP by solving its dual problem, Furthest Neighbor Search (FN). FN represents the problem that given a query $x \in \mathbb{R}^d$, we need to design a data structure to retrieve vector y from a dataset $Y \subset \mathbb{R}^d$ such that $\|x - y\|_2$ is minimized. Given the query set $X \subset \mathbb{R}^d$ and a dataset $Y \subset \mathbb{R}^d$, we perform the following transformations for any $x \in X$ and $y \in Y$.

$$\varphi(x) = \begin{bmatrix} x^\top / D_X & 0 & \sqrt{1 - \|x\|_2^2 / D_X^2} \end{bmatrix}^\top, \psi(y) = \begin{bmatrix} y^\top / D_Y & \sqrt{1 - \|y\|_2^2 / D_Y^2} & 0 \end{bmatrix}^\top,$$

where D_X is the maximum diameter of X and and D_Y is the maximum diameter of Y. In this way, we map $x \in X$ and $y \in Y$ to unit vectors. Moreover, $\|\varphi(x) - \psi(y)\|_2^2 = 2 - 2\langle \varphi(x), \psi(y)\rangle$. Furthermore, $\arg\max_{y \in Y} \|\varphi(x) - \psi(y)\|_2 = \arg\min_{y \in Y} \langle x, y\rangle$. Therefore, we could retrieve Min-IP of x with respect to Y by retrieving FN of $\varphi(x)$ with respect to $\psi(x)$. Therefore, our goal is to look for a data structure that solves FN efficiently. Note that the transformations should avoid the target minimum inner product to be negative, close to zero, or close to 1. Therefore, we may increase or decrease D_x and D_y accordingly. We remark such a formulation can be extended to various one-sided discrepancy-type problem, which progresses through upper barrier potential function.

Design of Data Structures. Though there exists available FN data structures, the integration of them to solve Min-IP problem in optimization is not straightforward. We design our FN data structure for Min-IP following these criterion: 1). the data structure should have theoretical guarantees on the query time, preprocessing time and space, 2). the space-time trade-offs of the data structure should adapt to the our problem, 3). the Min-IP estimation error by the data structure would not break the convergence of the optimization algorithm, 4). the data-structure should support dynamic operations such as insertion and deletion of point from the data structure, 5). the data-structure should handle non-independent Min-IP queries.

We start with the exact FN solution such as furthest-point Voronoi diagrams. Given a d-dimensional, n-point dataset, furthest-point Voronoi diagrams takes query time $O(d \log n)$ to retrieve the exact FN for a query $q \in \mathbb{R}^d$. However, the preprocessing time of furthest-point Voronoi diagrams is exponential to the dimension d, which makes the data structure unscalable to higher dimension. To overcome this curse of dimensionality, we focus on the approximate FN data structures with faster preprocessing time. In order to achieve a better overall runtime using approximate data structure, we need to consider the impact of using an approximate solution on the set S we construct. We show that if at each round, we can only find an index i^* such that the quantity of Eq. (2) is bounded by some $\beta > 1$, then the quality of our final set S is β times worse than the exact construction. More specifically, at each round, we query our approximate data structure to return an index i^* such that

$$\frac{v_{i^*}^\top (a_{j+1} I - T)^{-2} v_{i^*}}{\Phi^{a_j}(T) - \Phi^{a_{j+1}}(T)} + v_{i^*}^\top (a_{j+1} I - T)^{-1} v_{i^*} \leq \beta.$$

Fortunately, it is possible to modify the update rule to achieve this approximation, in the expense of worse quality of solution. Intuitively, since the criteria value has been blown up by a factor β, we want to compensate it through scaling down the outer product $v_{i^*} v_{i^*}^\top$ by a factor of β. Instead of directly updating the matrix T via $v_{i^*}v_{i^*}^\top$, we only use $\frac{1}{\beta}$ copies of this outer product to update T. By doing so, we can progress the greedy construction of S using this scaled-down version of T and finding a vector with β-approximate criteria at each iteration. The final result we obtain is on the spectral norm of matrix T, since overall, we scale it down by a factor of β, our final solution is β times worse than the exact algorithm:

$$\left\| \sum_{i \in S} v_i v_i^\top \right\| \leq \beta \cdot (\frac{n}{m} + O(\frac{1}{\sqrt{N}})).$$

We remark that utilizing data structure is especially valuable when the number of iterations n is large. Greedy algorithms without using data structure such as Alg. 1 and Alg. 2 don't need to pay for preprocessing/initialization time and space, in the expense of slower query time per iteration. In contrast, algorithms with data structure as Alg. 3 and Alg. 4 pay more time on preprocessing while enjoying a better query time. Note that the preprocessing time can be amortized into the cost per iteration via $\mathcal{T}_{\text{init}} + \#\text{iters} \cdot \mathcal{T}_{\text{query}} = \#\text{iters} \cdot (\frac{\mathcal{T}_{\text{init}}}{\#\text{iters}} + \mathcal{T}_{\text{query}})$, as $\#\text{iters} = n$ becomes larger, the preprocessing time becomes less relevant and the use of data structure becomes advantageous. This justifies our use of approximate data structure, which has a slightly large preprocessing time with a small query time. However, it is still unclear to us whether it is possible to get an exact data structure not suffering from the curse of dimensionality and hence becomes feasible under large iteration number n.

Our data structure should also support insertion and deletion of data points. The major reason is the search space ($[m] \setminus S$ in one-sided Kadison-Singer and S in experimental design) is dynamically changing, we need to add or remove new point in the process. Fortunately, with our AFN data structure, the time complexity of insertion and deletion data point is equal to the query time complexity. This property leads to efficient maintenance of data structures. However, in our formulation, the query to AFN data structure at each step is dependent on the previous steps. Thus, the standard union bound over queries could not be directly applied. We propose a query quantization approach that first locates the query into its nearest vertex on an ε-net. Then, we query with the vertex in the data structure. Although this operation would introduce an ε additive error on the final estimation of inner product, the total failure probability could be union bounded.

2.3 Application and Future Direction

Besides one-sided Kadison-Singer problem and experimental design problem, our framework can be further extended to one-sided discrepancy problem with a vanilla greedy construction using a minimum inner product criteria, similar to (2). Another surprising and interesting application is on cutting plane method (CPM). One key sub-task of a CPM solver is to find a row of constraint matrix A with small leverage score below certain threshold, which is exactly the kind of task we can apply $\mathsf{Min\text{-}IP}$ data structure. We give a preliminary view of such framework in Section A.

The major message of this paper is a general algorithmic framework of solving discrepancy problem with one-sided guarantee, using dynamic data structures. By preprocessing vectors, we build up $\mathsf{Min\text{-}IP}$ data structures with efficient query and update, which helps to reduce the cost per iteration in an iterative scheme. One of the ultimate goals along this line of work is to solve the algorithmic version of Kadison-Singer problem (two-sided), i.e., find a set S with the guarantee $\theta \leq \sum_{i \in S} |\langle u, v_i \rangle|^2 \leq s - \theta$, in quasi-polynomial time with respect to m. From high level, we ask the following data structure design problem: given m vectors v_1 through v_m, does there exist a data structure that can preprocess these m vectors and answer the query on the set of vectors $\sum_{i=1}^m c_i v_i$, where $c_i \in \{-1, +1\}$? Intuitively, given m points, can we design a data structure that answers query in a restricted span of these m points and can be dynamically updated, in quasi-polynomial time? We believe such data structure can be exploited to solve the original Kadison-Singer problem in quasi-polynomial time, which is better than current state-of-the-art involving approximating largest root of polynomial in $2^{\widetilde{O}(m^{1/3})}$ time [AGSS18]. Note that when $d = \mathrm{poly}(\log m)$, we can aim at a preprocessing time of $m^{\mathrm{poly}(d)} = 2^{\mathrm{poly}(\log m)}$ which could already be significant progress on algorithmic Kadison-Singer.

3 Literature Review: Data Structure and Optimization

We first give a summarization of this section. The two major directions we want to review is furthest-neighbor search and optimization with efficient data structure. The furthest-neighbor search problem is a well-studied data structure problem, but seeing much rare applications compared to the nearest-neighbor problem. We review the exact and approximate versions of it. Optimization with the use of data structure has been a popular trend these years. We review its applications in linear programming and convex programming.

3.1 Literature on Furthest Neighbor

Given a query vector $q \in \mathbb{R}^d$, the goal of the furthest neighbor problem is to retrieve a vector p from a n-point set $P \subset \mathbb{R}^d$ such that the $\|q - p\|_2$ is maximized. Over the last several decades, plenty of methods are proposed to solve the furthest neighbor problem without computing the distance with each element in the database. We categorized the existing literature into exact and approximate methods. We also discuss both algorithmic results and hardness results.

Exact Furthest Neighbor. For exact method, [Yao82] show that there this an algorithm that takes $O(d \log n)$ query time, $O(n^{2^{d+1}})$ preprocessing time and space to solve furthest neighbor problem. [VKSdBO00] constructs the furthest point Voronoi diagram to solve the furthest neighbor problem. The furthest point Voronoi diagram achieves query time $O(d \log n)$ with preprocessing time $O(n \log n + n^{d/2})$ and space $O(n^{d/2})$. [AMS92] proves that there exists a randomized algorithm so that for a query in $\mathbb{R}^d$, it takes expected total query time $O(n^{1-1/(d/2+o(1))})$ with both space and preprocessing $O(n^{d/2+o(1)})$ time to find the furthest neighbor for every $q \in Q$ from P. However, these methods suffer from the curse of dimensionality and could not scale to higher dimension.

Approximate Furthest Neighbor. The approximate methods relax the objective of the furthest neighbor problem with a multiplicative error $\overline{c} > 1$ and propose the approximate furthest neighbor ($\overline{c}$-AFN) problem. Given the furthest neighbor with Euclidean distance greater than r, the AFN targets at retrieving vectors with distance at least $r/\overline{c}$. Bespamyatnikh [Bes96] first proposes a fair split tree and solve $(1+\varepsilon)$-AFN in query time $O(1/\varepsilon^{d-1})$ and space $O(n^2)$ with preprocessing time $O(n^2)$. This query time is exponential to d so that the fair split tree approaches also suffers from the curse of dimensionality. In the following decades, there exists a set of approaches to tackle this bottleneck. Indyk [Ind00] proves that a $(1+\varepsilon)$-AFN problem over n-points set $P \subseteq \{0,1\}^d$ could be reduced to an approximate nearest neighbor search problem over an n-points set in $\{0,1\}^d$ with approximate factor $1 + \varepsilon/6$. This statement open the direction for solving AFN in high dimension using approximate nearest neighbor techniques. Following this direction, Goel, Indyk and Varadarajan [GIV01] propose a reduction algorithm that solves $(1+\varepsilon)$-AFN in query time $O(dn^{1/(1+\varepsilon)})$ with both preprocessing time and space in $O(dn^{1+1/(1+\varepsilon)})$. Goel, Indyk and Varadarajan [GIV01] also shows that $\sqrt{2}$-AFN could be solved in query time $O(1)$ with both preprocessing time and space in $O(dn)$. Indyk [Ind03] proposes an asymmetric embedding approach to remove the exponential dependence in dimension. For any $\delta > 0$, this method solves $(\bar{c} + \delta)$-AFN in query time $O(n^{1/\bar{c}^2} d \log n \log^2_{1+\delta} d)$ with both preprocessing time and space in $O(n^{1+1/\bar{c}^2} d \log n \log^2_{1+\delta} d)$. Agarwal and Sharathkumar [AS10] propose an algorithm using Blurred Ball Cover that answers $(\sqrt{2} + \varepsilon)$-AFN in query time $O((d/\varepsilon^3) \log(1/\varepsilon))$, space $O((d/\varepsilon^3) \log(1/\varepsilon))$ and preprocessing time $O(n \cdot (d/\varepsilon^2) \log(1/\varepsilon))$.

In theory, [ACW16] also shows results for bichromatic AFN. The goal of bichromatic AFN is to find AFN for every element in the n-point query set $Q \subset \mathbb{R}^d$ from the n-point data set $P \subset \mathbb{R}^d$. [ACW16] proves that we could find $(1 + \varepsilon)$-AFN for each $q \in Q$ from P in total query time near $dn - n^{2-\Omega(\varepsilon^{1/3}/\log(1/\varepsilon))}$.

Constant approximation for Min-IP. In [ACW16], they showed an $n^{2-1/\widetilde{O}(\sqrt{c})}$ time algorithm that provides constant approximation for Min-IP with n vectors from $\{0,1\}^{c\log n}$. Later, Chen and Williams [CW19] showed an $n^{2-1/O(\log c)}$ time algorithm that gives constant approximation to Min-IP. Orthogonal vectors OV is conjectured to be no truly subquadratic time algorithm. [CW19] proved OV is equivalent to approximate Min-IP.

3.2 Previous Techniques for Speedup Optimization

The running time of a general iterative type (e.g., optimization) algorithm has two parts, one part is the number of iterations and the other part is the cost spent in each iteration. Over the last few decades, the number of iterations for many problems has been optimized. The next natural direction is to reduce the cost per iteration. In order to improve the running time of solvers for solvers such as central path method for linear programming (LP) and cutting plane method (CPM) for convex programming, a number of techniques such as sampling, sketching, vector-maintenance, sparse recovery have been proposed and studied. We will briefly survey all of these techniques in the following discussion. We also remark that the technique we use in this paper follows the trending direction in this field, which is to use efficient data structure to reduce the cost per iteration. However, the data structure we consider is novel and completely different from all these previous works.

Sampling. One popular method of solving linear programming is so-called central path method. Roughly speaking, it can be formulated as follows: given a fixed matrix A and two dynamic sequence of positive vectors $w^1, w^2, \cdots w^T \in \mathbb{R}^n$, and $h^1, h^2, \cdots, h^T \in \mathbb{R}^n$ let $P(w)$ be defined as $P(w) := \sqrt{W} A (A^\top W A)^{-1} A^\top \sqrt{W}$, where W is the diagonal matrix such that the i-th diagonal $W_{i,i}$ is w_i. The goal is to compute $P(w^t) \cdot h^t$ in each iteration. The most naive solution is to compute $P(w^t)$ and the following matrix-vector product at each iteration, but it is expensive and does not exploit the structure of the sequence w^t and h^t. One natural idea is to consider approximating the product $P(w^t) \cdot h^t$ instead of computing it exactly, if one can show that the general optimization framework can tolerate the error incurred by approximation, then the cost of computing both $P(w^t)$ and the matrix-vector product can hopefully be reduced. In a recent work [CLS19], Cohen, Lee and Song showed that it suffices to sample $\sqrt{n}$ coordinates of h^t to obtain a good error bound. More concretely, let $D \in \mathbb{R}^{n \times n}$ be the diagonal sampling matrix with roughly $\sqrt{n}$ non-zero entries. Applying D to h^t, they instead compute a matrix-vector product with much smaller support of the vector, which greatly improved the running time of this product. Combined with a novel projection maintenance technique, they delivered an algorithm that solves linear programming in time $\widetilde{O}(n^\omega + n^{2+1/6})$.

Sketching. Different from sampling, sketching is another powerful technique for saving cost on matrix-vector product. Sketching has many applications in both TCS and ML problems [Sar06, CW13, NN13, BWZ16, SWZ17, ALS+18, XZZ18, SWZ19, WZD+20]. Recently, [LSZ19, SY21] showed how to use sketching matrix $R \in \mathbb{R}^{\sqrt{n} \times n}$ to recover the $\widetilde{O}(n^\omega + n^{2+1/6})$ result from [CLS19] via using sketching matrix R throughout the iterations. Mathematically speaking, they use the sketching matrix as follows: $R^\top R \cdot P(w)h$ and $P(w) \cdot R^\top R \cdot h$. The first one is typically called "sketch on the left", the intuition is the data structure maintains a sketched projection $R \cdot P(w^t)$ at each round, when querying the matrix-vector product $P(w^t) \cdot h^t$, the running time reduces from $O(n^2)$ to $O(n^{3/2})$. The second one can be viewed as "sketch on the right", where the vector h is sketched, this transforms a product between $n \times n$ matrix and length n vector into a chain product of $(P(w^t) \cdot R^\top) \cdot (Rh^t)$. By maintaining $P(w^t) \cdot R^\top$, the cost of product can be reduced to $O(n^{3/2})$.

Vec-maintenance. In the above sampling and sketching techniques, both algorithms treat vector h^t as worst case vector in each iteration, i.e., the change of coordinates from h^t and h^{t+1} can be large. However, [Bra20] has an elegant observation that the sequence $\{h^t\}_{t=1}^T$ is changing slowly in each iteration. Thus, maintaining the product $P(w^t) \cdot h^t$ or even part of it provides extra information and enables fast update. By doing so, [Bra20] is also able to recover the result in [CLS19] with a completely deterministic algorithm. In a more recent work [JSWZ21], they combined sketching and vector-maintenance to further improve the running time of linear program solver to $\widetilde{O}(n^\omega + n^{2+1/18})$.

Sparse Recovery. Another useful technique is to use sparse recovery, as illustrated in [BLSS20]. In their work, they formulate it as an abstract vector maintenance problem: given a sequence $\{h^t\}_{t=1}^T$ and $\{g^t\}_{t=1}^T$, maintain the sum $\sum_{i=1}^t G^i A h^i$ for some fixed conforming matrix A. They have shown that it is enough to get a multiplicative approximation of this sum, so it suffices to detect entries with large change across the iterations, and update those entries. Such a task can be combined with the well-studied heavy hitter/sparse recovery technique, which involves using a small, sparse and random sensing matrix $\Phi \in \mathbb{R}^{k \times n}$ with $k \ll n$ and "sensing" the large entries of vector x through Φx. Thus, the idea is to store $\Phi G^t A$ at each round, which can be computed fast if g^t does not change too much from g^{t-1}, and detect large entries through computing $(\Phi G^t A) h^t$. Joint with an inverse maintenance data structure that uses a sparsifier, they can solve linear program with a tall constraint matrix where there are d variables and n constraints and $n \gg d$, in time $\widetilde{O}(nd + d^3)$.

From LP to Cutting Plane Method. Cutting plane method is a class of optimization algorithms that iteratively queries a separation oracle to cut the feasible set that contains the optimal solution. There has been a long line of work to obtain fast cutting plane methods [Sho77, YN76, Kha80, KTE88, NN89, Vai89, AV95, BV02, LSW15, JLSW20].

The general framework of cutting plane method is similar to LP in the sense that it needs to maintain the projection $P(w^t)$, however, it is harder in the sense that the constraint matrix A is no longer fixed, at each round, a row of A is either inserted or deleted based on its leverage score, which corresponds to diagonal entries of $P(w^t)$. One can think of leverage score as a measurement of the importance of a constraint in matrix A, intuitively, if one constraint is "similar" to most other constraints, then it would have a small leverage score and thus be removed. Hence, the key challenge is to maintain the projection under insertion and deletion, and efficiently approximate leverage score. By using a three-layered data structure and sketching, [JLSW20] achieves a cutting plane method solver with $O(n \log \kappa)$ calls to separation oracle and an extra $O(n^2)$ cost per call, where $\kappa = nR/\varepsilon$ and R is the radius of the box.

Roadmap. In Section 4, we give a preliminary on notations, definitions, some useful facts and the problem setup for Kadison-Singer. In Section 5, we present our implementation of exact and approximate $\mathsf{Min\text{-}IP}$ data structure using furthest-neighbor search. In Section 6, we present our algorithmic result for one-sided Kadison-Singer problem with approximate guarantee. In Section 7, we utilize our algorithmic framework on the rounding up of experimental design problem. In Section A, we give a preliminary overview on how to use $\mathsf{Min\text{-}IP}$ data structure to implement for a sub-task of cutting plane method.

4 Preliminaries

This section gives some preliminary background definitions and facts.

- In Section 4.1, we introduce notations used across this paper.
- In Section 4.2, we list some useful facts for our later proof.

4.1 Notations

We introduce some notations and definitions we will use throughout this paper.

For a positive integer n, we use $[n]$ to denote the set $\{1, 2, \cdots, n\}$. For a vector x, we use $\|x\|_2$ to denote its ℓ_2 norm. For a matrix A, we use $\|A\|$ to denote its spectral norm. For a square matrix A, we use $\mathrm{tr}[A]$ to denote its trace. For a square and full rank matrix A, we use A^{-1} to denote its inverse.

We say a symmetric matrix $A \in \mathbb{R}^{n \times n}$ is positive semi-definite (PSD, denoted as $A \succeq 0$) if for any vector $x \in \mathbb{R}^n$, $x^\top A x \geq 0$. We say a symmetric matrix $A \in \mathbb{R}^{n \times n}$ is positive definite (PD, denoted as $A \succ 0$) if for any vector $x \in \mathbb{R}^n$, $x^\top A x > 0$.

For a real positive semi-definite matrix A, we define its square root $A^{1/2}$ to be the unique positive semi-definite matrix such that $(A^{1/2})^\top A^{1/2} = A$.

For two conforming matrices A and B, we have $\mathrm{tr}[AB] = \mathrm{tr}[BA]$.

For a real symmetric matrix A, we use $\lambda_{\max}(A)$ to denote its largest eigenvalue and $\lambda_{\min}(A)$ to denote its smallest eigenvalue.

We define $\mathcal{T}_{\mathrm{mat}}(a, b, c)$ to be the time of multiplying an $a \times b$ matrix with another $b \times c$ matrix. Note that $\mathcal{T}_{\mathrm{mat}}(a, b, c) = O(\mathcal{T}_{\mathrm{mat}}(a, c, b)) = O(\mathcal{T}_{\mathrm{mat}}(b, a, c))$.

4.2 Useful Facts

We list and prove some useful facts regarding matrices.

Fact 4.1. *For any PSD matrix $Z \in \mathbb{R}^{d \times d}$, we have $\mathrm{tr}[Z^{1/2}] \leq \sqrt{d \cdot \mathrm{tr}[Z]}$.*

Proof. Note that for any $d \times d$ positive semi-definite matrix $Z \succeq 0$, $\mathrm{tr}[Z^{1/2}] \leq \sqrt{d \cdot \mathrm{tr}[Z]}$ due to Cauchy-Schwartz inequality applied to the non-negative spectrum of $Z^{1/2}$. $\square$

Fact 4.2 (matrix Woodbury identity, [Woo49, Woo50])**.** *For matrices $M \in \mathbb{R}^{n \times n}$, $U \in \mathbb{R}^{n \times d}$, $C \in \mathbb{R}^{d \times d}$, $V \in \mathbb{R}^{d \times n}$,*

$$(M + UCV)^{-1} = M^{-1} - M^{-1}U(C^{-1} + VM^{-1}U)^{-1}VM^{-1}.$$

Fact 4.3. *Let A and B denote two diagonal matrices in $\mathbb{R}^{d \times d}$. Suppose $\forall i \neq j \in [n]$, we have $\beta_i - \alpha_i = \beta_j - \alpha_j$, and let $\gamma = \beta_i - \alpha_i$. We have*

$$\mathrm{tr}[A^{-1} - B^{-1}] = \gamma \cdot \mathrm{tr}[A^{-1}B^{-1}].$$

Proof. We have

$$\begin{aligned}\mathrm{tr}[A^{-1} - B^{-1}] &= \sum_{i=1}^{k} \frac{1}{\alpha_i} - \frac{1}{\beta_i} \\ &= \sum_{i=1}^{k} \frac{\beta_i - \alpha_i}{\alpha_i \beta_i} \\ &= \gamma \sum_{i=1}^{k} \frac{1}{\alpha_i \beta_i} \\ &= \gamma \cdot \mathrm{tr}[A^{-1} B^{-1}]\end{aligned}$$

Thus, we complete the proof. $\square$

Fact 4.4 (Inequality for two monotone sequences)**.** *Suppose* $a_1 \geq a_2 \geq \cdots \geq a_n \geq 0$*,* $b_1 \geq \cdots \geq b_n \geq 0$*, then we have*

$$\sum_{i=1}^{n} a_i b_{n-i} \leq \frac{1}{n} \sum_{i=1}^{n} a_i \sum_{j=1}^{n} b_j$$

5 Data Structures

In this section, we give an overview of the data structures we will be using. Specifically, we implement the key Min-IP data structure via furthest-neighbor search.

- In Section 5.1, we give an exact implementation of Min-IP via Voronoi diagram.
- In Section 5.2, we discuss randomized approximate furthest-neighbor search data structure.
- In Section 5.3, we implement approximate Min-IP via approximate furthest-neighbor search.
- In Section 5.4, we show how to transform a non-unit vector into unit through padding extra dimensions.
- In Section 5.5, we illustrate how to handle adaptive adversaries in our data structure.
- In Section 5.6, we generalize our data structure to handle complex vectors.

5.1 Exact Min-IP

In this section, we implement exact Min-IP data structure via Voronoi diagram, with a fast query and update time in the expense of exponential preprocessing time.

Definition 5.1 (Furthest-Neighbor (FN))**.** Given an n-point dataset $P \subset \mathbb{S}^{d-1}$ on the sphere, the goal of the Furthest-Neighbor (FN) problem is to build a data structure that, given a query $q \in \mathbb{S}^{d-1}$, retrieve the solution of $\arg\max_{p \in P} \|p - q\|_2$.

Definition 5.2 (Min-IP)**.** Given an n-point dataset $P \subset \mathbb{S}^{d-1}$ on the sphere, the goal of the Minimum Inner Product Search (Min-IP) is to build a data structure that, given a query $q \in \mathbb{S}^{d-1}$, retrieve the solution of $\arg\min_{p \in P} \langle p, q \rangle$.

For any two points x, y with $\|x\|_2 = \|y\|_2 = 1$, we have $\|x - y\|_2^2 = 2 - 2\langle x, y\rangle$. Thus, Min-IP (Definition 5.2) is equivalent to FN (Definition 5.1).

Inspired by [VKSdBO00], we have the following Theorem.

Theorem 5.3. *The* Min-IP *(Definition 5.2) could be solved by a data-structure in query time $O(d \log n)$ and space $O(n^{d/2})$ with preprocessing time $O(n \log n + n^{d/2})$. Moreover, the data-structure support deletion and insertion of new data vector in $O(d \log n)$ time.*

Proof. From [VKSdBO00], we know that we could solve FN (Definition 5.1) with furthest-point Voronoi diagram in query time $O(d \log n)$, space $O(n^{d/2})$ and preprocessing time $O(n \log n + n^{d/2})$. As the solution of FN is also the solution of Min-IP. We finish the proof. $\square$

5.2 Approximate Furthest-Neighbor

In this section, we implement approximate Min-IP via AFN data structure introduced by Indyk [Ind03]. It has slightly slower query time compared to Voronoi diagram, but the preprocessing does not suffer from curse of dimensionality.

Definition 5.4 (Approximate Furthest-Neighbor (AFN))**.** Let $\bar{c} > 1$ and $r \in (0, 2)$. Given an n-point dataset $P \subset \mathbb{S}^{d-1}$ on the sphere, the goal of the $(\bar{c}, r)$-Approximate Furthest-Neighbor (AFN) problem is to build a data structure that, given a query $q \in \mathbb{S}^{d-1}$ with the promise that there exists a point $p \in P$ with $\|p - q\|_2 \geq r$ reports a point $p' \in P$ with distance $\|p - q\|_2 \geq r/\bar{c}$.

Definition 5.5 (Approximate Min-IP)**.** Let $c \in (0, 1)$ and $\tau \in (0, 1)$. Given an n-point dataset $P \subset \mathbb{S}^{d-1}$ on the sphere, the goal of the (c, τ)-Minimum Inner Product Search (Min-IP) is to build a data structure that, given a query $q \in \mathbb{S}^{d-1}$ with the promise that there exists a point $p \in P$ with $\langle p, q\rangle \leq \tau$, it reports a point $p' \in P$ with similarity $\langle p', q\rangle \leq \tau/c$.

Theorem 5.6 ([Ind03])**.** *Let $\bar{c} > 1$ and $r \in (0, 2)$. For any $\delta > 0$, the $(\bar{c} + \delta, r)$-*AFN *could be solved by a data structure in query time $O(n^{1/\bar{c}^2} d \log n \log^2_{1+\delta} d)$*[2] *and space $O(n^{1+1/\bar{c}^2} d \log n \log^2_{1+\delta} d)$ with preprocessing time $O(n^{1+1/\bar{c}^2} d \log n \log^2_{1+\delta} d)$. Moreover, the data structure supports deletion and insertion of new point in $O(n^{1/\bar{c}^2} d \log n \log^2_{1+\delta} d)$ time. For any fixed query, the success probability is* 0.99.[3]

5.3 Min-IP via AFN

For any two points x, y with $\|x\|_2 = \|y\|_2 = 1$, we have $\|x - y\|_2^2 = 2 - 2\langle x, y\rangle$. This implies that $r^2 = 2 - 2\tau$. Further,for any $\delta > 0$, if we have a data structure for $(\bar{c} + \delta, r)$-AFN , it automatically becomes a data structure for (c, τ)-Min-IP with parameters $\tau = 1 - 0.5r^2$ and $c = \frac{1 - 0.5r^2}{1 - 0.5r^2/(\bar{c}+\delta)^2}$. This implies that

$$\begin{aligned}(\bar{c} + \delta)^2 &= \frac{cr^2}{2c - 2 + r^2} \\ &= \frac{c(2 - 2\tau)}{2c - 2 + (2 - 2\tau)}\end{aligned}$$

[2]The actual dependence is $O(n^{1/\bar{c}^2} d \log^{1 + \frac{1 - \bar{c}^2}{2}} n \log_{1+\delta} d \log \log_{1+\delta} d)$. In [PSSS15], they improved a logarithmic factor by modifying the data structure of [Ind03]. In our applications, we do not care about log factors, thus we simplify the bound.

[3]In our applications, we will use data structure with boosted success probability (see page 4 of [Ind03]). We will show how to perform such boosting in Section 5.5.

$$= \frac{c(1-\tau)}{c-\tau} \tag{4}$$

Next, we show that $\frac{c(1-\tau)}{c-\tau}$ is increasing as τ increase and $\frac{c(1-\tau)}{c-\tau} > 8$ if $\tau \in (\frac{7c}{8-c}, c)$.

Lemma 5.7. *Let $c \in (0,1)$ and $\tau \in (0,1)$, we show that function $f(c,\tau) := \frac{c(1-\tau)}{c-\tau}$ is decreasing as c increase and increasing as τ increase.*

Proof. We take the derivative of $f(c,\tau)$ over c and get

$$\frac{\partial}{\partial c} f(c,\tau) = \frac{(\tau-1)\tau}{(c-\tau)^2} < 0$$

where the second step follows from $c > \tau$ and $\tau < 1$.

Therefore, $f(c,\tau) := \frac{c(1-\tau)}{c-\tau}$ is decreasing as c increase.

We take the derivative of $f(c,\tau)$ over τ and get

$$\begin{aligned} \frac{\partial}{\partial \tau} f(c,\tau) &= \frac{c(\tau^2 - 2c\tau + c)}{(c-\tau)^2} \\ &= \frac{c((\tau-c)\tau + c(1-\tau))}{(c-\tau)^2} > 0 \end{aligned}$$

where the second step follows from $c > \tau$ and $\tau < 1$.

Therefore, $f(c,\tau) := \frac{c(1-\tau)}{c-\tau}$ is increasing as τ increase.

□

Next, we show the query time, space and preprocessing time to solve (c,τ)-Min-IP using Theorem 5.6.

Corollary 5.8. *Let $\tau \in (0,1)$. If we set $c \in (\tau, \frac{8\tau}{\tau+7})$, then the (c,τ)-Min-IP on a unit sphere $\mathbb{S}^{d-1}$ could be solved in query time $O(n^{0.5} d \log^3 n)$ and space $O(n^{1.5} d \log^3 n)$ with preprocessing time $O(n^{1.5} d \log^3 n)$.*

Proof. We know that if we let $\delta = \overline{c}$ and have a data structure for $(2\overline{c}, r)$-AFN , it automatically becomes a data structure for (c,τ)-Min-IP with parameters $\tau = 1 - 0.5r^2$ and $c = \frac{1-0.5r^2}{1-0.5r^2/4\overline{c}^2}$. Moreover, using Eq.(4), we have

$$\begin{aligned} \overline{c}^2 &= \frac{c(1-\tau)}{4(c-\tau)} \\ &= \frac{c(1-\tau)}{4(c-\tau)} \end{aligned} \tag{5}$$

if $\tau \in (0,1)$ and $c \in (\tau, \frac{8\tau}{\tau+7})$, we have

$$\begin{aligned} \overline{c}^2 &= \frac{c(1-\tau)}{4(c-\tau)} \\ &> \frac{8\tau}{\tau+7} \cdot \frac{1-\tau}{4(\frac{8\tau}{\tau+7} - \tau)} \\ &= \frac{8\tau}{\tau+7} \cdot \frac{(1-\tau)(\tau+7)}{4(8\tau - \tau^2 - 7\tau)} = 2 \end{aligned}$$

Next, using Theorem 5.6, we show that the $(2\overline{c}, r)$-AFN could be solved with query time $O(n^{1/\overline{c}^2} d \log n \log^2_{1+\overline{c}} d)$, proprocessing time/space $O(n^{1+1/\overline{c}^2} d \log n \log^2_{1+\overline{c}} d)$.

Combining the Theorem 5.6 with $\overline{c}^2 > 2$, we rewrite $n^{1/\overline{c}^2} d \log n \log^2_{1+\overline{c}} d$ as

$$n^{1/\overline{c}^2} d \log n \log^2_{1+\overline{c}} d < n^{1/\overline{c}^2} d \log n \log^2 d < n^{0.5} d \log n \log^2 d \le n^{0.5} d \log^3 n$$

where the first step follows from $\overline{c} > 1$ and the second step follows from $\overline{c}^2 > 2$, the third step follows from $n \ge d$.

In this way, we could rewrite query time as $O(n^{0.5} d \log^3 n)$. Similarly, we could rewrite both preprocessing time and space as $O(n^{1.5} d \log^3 n)$. Thus, we finish the proof. $\square$

Corollary 5.9. *Let $\tau \in (0,1)$. If we set $c \in (\tau, \frac{40\tau}{\tau+39})$, then the (c,τ)-$\mathsf{Min\text{-}IP}$ on a unit sphere $\mathbb{S}^{d-1}$ could be solved in query time $O(n^{0.1} d \log^3 n)$ and space $O(n^{1.1} d \log^3 n)$ with preprocessing time $O(n^{1.1} d \log^3 n)$.*

Proof. From Eq. (5), we know that

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

if $\tau \in (0,1)$ and $c \in (\tau, \frac{40\tau}{\tau+39})$, we have

$$\begin{aligned}\overline{c}^2 &= \frac{c(1-\tau)}{4(c-\tau)} \\ &> \frac{40\tau}{\tau+39} \cdot \frac{1-\tau}{4(\frac{40\tau}{\tau+39} - \tau)} \\ &= \frac{40\tau}{\tau+39} \cdot \frac{(1-\tau)(\tau+39)}{4(40\tau - \tau^2 - 39\tau)} = 10\end{aligned}$$

where the second step follows from Lemma 5.7 that $\frac{c(1-\tau)}{c-\tau}$ is decreasing as c increase, the third and forth step are reorgnizations.

Next, using Theorem 5.6, we show that the $(2\overline{c}, r)$-AFN could be solved with query time $O(n^{1/\overline{c}^2} d \log n \log^2_{1+\overline{c}} d)$, proprocessing time/space $O(n^{1+1/\overline{c}^2} d \log n \log^2_{1+\overline{c}} d)$.

Combining the Theorem 5.6 with $\overline{c}^2 > 10$, we rewrite $n^{1/\overline{c}^2} d \log n \log^2_{1+\overline{c}} d$ as

$$n^{1/\overline{c}^2} d \log n \log^2_{1+\overline{c}} d < n^{1/\overline{c}^2} d \log n \log^2 d < n^{0.1} d \log n \log^2 d \le n^{0.1} d \log^3 n$$

where the first step follows from $\overline{c} > 1$ and the second step follows from $\overline{c}^2 > 10$, the third step follows from $n \ge d$.

In this way, we could rewrite query time as $O(n^{0.1} d \log^3 n)$. Similarly, we could rewrite both preprocessing time and space as $O(n^{1.1} d \log^3 n)$. Thus, we finish the proof. $\square$

Corollary 5.10. *Let $\tau \in (0,1)$. If we set $c \in (\tau, \frac{400\tau}{\tau+399})$, then the (c,τ)-$\mathsf{Min\text{-}IP}$ on a unit sphere $\mathbb{S}^{d-1}$ could be solved in query time $O(n^{0.01} d \log^3 n)$ and space $O(n^{1.01} d \log^3 n)$ with preprocessing time $O(n^{1.01} d \log^3 n)$.*

Proof. From Eq. (5), we know that

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

if $\tau \in (0,1)$ and $c \in (\tau, \frac{400\tau}{\tau+399})$, we have

$$\begin{aligned}
\overline{c}^2 &= \frac{c(1-\tau)}{4(c-\tau)} \\
&> \frac{400\tau}{\tau+399} \cdot \frac{1-\tau}{4(\frac{400\tau}{\tau+399}-\tau)} \\
&= \frac{400\tau}{\tau+399} \cdot \frac{(1-\tau)(\tau+399)}{4(400\tau-\tau^2-399\tau)} = 100
\end{aligned}$$

where the second step follows from Lemma 5.7 that $\frac{c(1-\tau)}{c-\tau}$ is decreasing as c increase, the third and forth step are reorgnizations.

Next, using Theorem 5.6, we show that the $(2\overline{c}, r)$-AFN could be solved with query time $O(n^{1/\overline{c}^2} d \log n \log^2_{1+\overline{c}} d)$, preprocessing time/space $O(n^{1+1/\overline{c}^2} d \log n \log^2_{1+\overline{c}} d)$.

Combining the Theorem 5.6 with $\overline{c}^2 > 100$, we rewrite $n^{1/\overline{c}^2} d \log n \log^2_{1+\overline{c}} d$ as

$$n^{1/\overline{c}^2} d \log n \log^2_{1+\overline{c}} d < n^{1/\overline{c}^2} d \log n \log^2 d < n^{0.01} d \log n \log^2 d \leq n^{0.01} d \log^3 n$$

where the first step follows from $\overline{c} > 1$ and the second step follows from $\overline{c}^2 > 100$, the third step follows from $n \geq d$.

In this way, we could rewrite query time as $O(n^{0.01} d \log^3 n)$. Similarly, we could rewrite both preprocessing time and space as $O(n^{1.01} d \log^3 n)$. Thus, we finish the proof. □

5.4 Transformations

In most applications, query and data vectors are usually not unit vectors. Therefore, we need to transform them into unit vectors without breaking the Min-IP solution. We propose a pair of asymmetric transformations as below:

Definition 5.11. Given the query set $X \subset \mathbb{R}^d$ and a dataset $Y \subset \mathbb{R}^d$, we performs the following transformations for any $x \in X$ and $y \in Y$.

$$\varphi(x) = \begin{bmatrix} \frac{x^\top}{D_X} & 0 & \sqrt{1 - \frac{\|x\|_2^2}{D_X^2}} \end{bmatrix}^\top, \psi(y) = \begin{bmatrix} \frac{y^\top}{D_Y} & \sqrt{1 - \frac{\|y\|_2^2}{D_Y^2}} & 0 \end{bmatrix}^\top,$$

where D_X is the maximum diameter of X and and D_Y is the maximum diameter of Y. In this way, we map $x \in X$ and $y \in Y$ to unit vectors. Moreover, $\|\varphi(x) - \psi(y)\|_2^2 = 2 - 2\langle \varphi(x), \psi(y) \rangle$ and $\arg\max_{y \in Y} \|\varphi(x) - \psi(y)\|_2 = \arg\min_{y \in Y} \langle x, y \rangle$.

Remark 5.12. In our later applications, we implicitly assume all points have undergone such transformations in preprocessing phase. We also remark that in query phase, the set Y consists of a single query point, it suffices to pick D_Y as $\|y\|_2$, in this case, the transformation can be viewed as normalizing the query vector. If computing the inner product between x and y is required, we can retrieve the original x and y by its first d dimension, and by storing D_X as a variable in the data structure.

5.5 Adaptive Queries

The queries at each step for Min-IP are not independent. Therefore, we could not direct union bound the failure probability. We propose a quantize method to handle this problem. For each query $q \in \mathbb{R}^d$, we first quantize it into vertex $\widehat{q}$ in the ε-net. Then, we use $\widehat{q}$ as query. Although this would generate an ε additive error, we could bound the failure probability of a series of dependent queries.

Given a query set $Q \subseteq \mathbb{R}^d$ with maximum diameter D and a n-point dataset $P \subset \mathbb{R}^d$, the failure probability of a series of dependent queries from $Q \subset \mathbb{R}^d$ is equal to the probability that at least on vertex on the ε-net of Q fails as a query, which is written as $n\widetilde{\delta} \cdot (\frac{dD}{\varepsilon})^d$, where $\widetilde{\delta}$ is the failure probability of the data structure. We could also reduce the failure probability by repeating the data structure for l times, which will reduce the total failure probability to $n\widetilde{\delta}^l \cdot (\frac{dD}{\varepsilon})^d$ but increase the query complexity by l times.

Remark 5.13. It is standard to reduce the constant failure probability to $\widetilde{\delta}$ by repeating the data-structure $\log(1/\widetilde{\delta})$ copies. In addition, we also need to deal with adaptive query, therefore we need to choose $\widetilde{\delta} = \delta/n^d$ to sufficiently to union all the points in the ε-net. Thus we blow up the time by an extra $d\log(n/\delta)$ factor with paying δ failure probability (after union bound over all iterations).

We also remark that by picking $\varepsilon = \frac{\tau}{c}$ and using $d\log(n/\delta)$ copies of independent data structures, we obtain a collection of data structures with success probability at least $1-\delta$ and the guarantee of inner product transforms from $\langle p, q\rangle \le \frac{\tau}{c}$ to $\langle p', q\rangle \le 2 \cdot \frac{\tau}{c}$.

5.6 Complex Vectors

In our problem, the queries and data vectors could be complex vectors. Therefore, we need to adapt the data structures for complex-valued inputs. We achieve this through transforming a complex vector into a real vector with twice the length, where the first half of the values are real part and the second half of are complex part. This is a standard practice to extend data structures for complex-valued vectors.

6 One-Sided Kadison-Singer via Furthest-Neighbor Search

In this section, we provide efficient data structure for one-sided Kadison-Singer problem.

- In Section 6.1, we state some useful facts and lemmas.
- In Section 6.2, we formally define the one-sided Kadison-Singer problem.
- In Section 6.3, we prove the correctness of a greedy process with an approximate guarantee.
- In Section 6.4, we give an overview of algorithms and compare their running time.
- In Section 6.5, we provide an analysis for a straightforward implementation of the greedy process.
- In Section 6.6, we give a slightly better implementation.
- In Section 6.7, we introduce Voronoi diagram with fast query time but suffers from curse of dimensionality.
- In Section 6.8, we use approximate Min-IP data structure to achieve both fast preprocessing and query time.

6.1 Useful Facts

In this section, we introduce some useful facts and lemmas to facilitate the proof of the core approximate greedy lemma.

Definition 6.1 (Upper barrier potential)**.** Let T denote a square matrix. For any $a > \|T\|$, we define potential function $\Phi^a(T)$ as follows

$$\Phi^a(T) := \mathrm{tr}[(aI - T)^{-1}].$$

This potential function has been studied and popularized in [Sri10, BSS12]. We state a tool from previous [Sri10](see Lemma 3.4), for the completeness, we still provide a proof.

Lemma 6.2. *For any a, δ and $T \in \mathbb{C}^{d \times d}$ satisfying $\|T\| < a$, and for any $t \geq 0$, if*

$$\frac{v^\top((a+\delta)I - T)^{-2}v}{\Phi^a(T) - \Phi^{a+\delta}(T)} + v^\top((a+\delta)I - T)^{-1}v \leq \frac{1}{t},$$

Then

- $\Phi^{a+\delta}(T + t \cdot vv^\top) \leq \Phi^a(T)$,
- $\|T + t \cdot vv^\top\| < a + \delta$.

Proof. Let $\widetilde{u} = a + \delta$ and

$$U := \frac{v^\top((a+\delta)I - T)^{-2}v}{\Phi^a(T) - \Phi^{a+\delta}(T)} + v^\top((a+\delta)I - T)^{-1}v.$$

Part 1. By the Sherman-Morrison formula, we can write the updated potential as:

$$\begin{aligned}
\Phi^{\widetilde{u}}(T + tvv^\top) &= \mathrm{tr}[(\widetilde{u}I - T - tvv^\top)^{-1}] \\
&= \mathrm{tr}[(\widetilde{u}I - T)^{-1} + \frac{t \cdot (\widetilde{u}I - T)^{-1}vv^\top(\widetilde{u}I - T)^{-1}}{1 - tv^\top(\widetilde{v}I - T)^{-1}v}] \\
&= \mathrm{tr}[(\widetilde{u}I - T)^{-1}] + \frac{t \cdot \mathrm{tr}[v^\top(\widetilde{u}I - T)^{-1}(\widetilde{u}I - T)^{-1}v]}{1 - t \cdot v^\top(\widetilde{u}I - T)^{-1}v} \\
&= \Phi^{\widetilde{u}}(T) + \frac{tv^\top(\widetilde{u}I - T)^{-2}v}{1 - t \cdot v^\top(\widetilde{u}I - T)^{-1}v} \\
&= \Phi^a(T) - (\Phi^a(T) - \Phi^{\widetilde{u}}(T)) + \frac{tv^\top(\widetilde{u}I - T)^{-2}v}{1 - t \cdot v^\top(\widetilde{u}I - T)^{-1}v} \\
&= \Phi^a(T) - (\Phi^a(T) - \Phi^{\widetilde{u}}(T)) + \frac{v^\top(\widetilde{u}I - T)^{-2}v}{1/t - v^\top(\widetilde{u}I - T)^{-1}v}
\end{aligned}$$

where the second step follows from Sherman-Morrison formula, the third step follows from $\mathrm{tr}[\cdot]$ is linear and $\mathrm{tr}[XY] = \mathrm{tr}[YX]$.

Note that $U \leq \frac{1}{t}$, this means that

$$\begin{aligned}
&\frac{v^\top(\widetilde{u}I - T)^{-2}v}{1/t - v^\top(\widetilde{u}I - T)^{-1}v} - (\Phi^a(T) - \Phi^{\widetilde{u}}(T)) \\
\leq\ &\frac{v^\top(\widetilde{u}I - T)^{-2}v}{U - v^\top(\widetilde{u}I - T)^{-1}v} - (\Phi^a(T) - \Phi^{\widetilde{u}}(T))
\end{aligned}$$

$$= \frac{v^\top(\widetilde{u}I - T)^{-2}v}{\frac{v^\top(\widetilde{u}I-T)^{-2}v}{\Phi^a(T)-\Phi^{\widetilde{u}}(T)} + v^\top(\widetilde{u}I - T)^{-1}v - v^\top(\widetilde{u}I - T)^{-1}v} - (\Phi^a(T) - \Phi^{\widetilde{u}}(T))$$
$$= 0$$

This shows that $\Phi^{\widetilde{u}}(T + tvv^\top) \leq \Phi^a(T)$.

Part 2. For $\|T + tvv^\top\|$, suppose this is not true, then there exists some $t' > 0$ such that $\|T+t'vv^\top\| = \widetilde{u}$. In this case, at least one eigenvalue of $\widetilde{u}-(T+t'vv^\top)$ is 0, this means $\Phi^{\widetilde{u}}(T+tvv^\top)$ will be infinite. However, this is not possible, since $\Phi^{\widetilde{u}}(T)$ is finite. The term

$$\frac{v^\top(\widetilde{u}I - T)^{-2}v}{1/t - v^\top(\widetilde{u}I - T)^{-1}v}$$

is also finite due to the positive definiteness of matrix $(\widetilde{u}I - T)^{-2}$ and $\Phi^a(T) > \Phi^{\widetilde{u}}(T)$. Therefore, $U > v^\top(\widetilde{u}I - T)^{-1}v$, and the term $1/t - v^\top(\widetilde{u}I - T)^{-1}v$ is nonzero.

This establishes the fact that $\|T + tvv^\top\| < a + \delta$. □

6.2 Problem Setup

We consider a version of Weaver's discrepancy theoretical formulation of Kadison-Singer problem, introduced in [Wea13].

Question 6.3. *Does there exist a constant N and r, such that if $\{v_1, \ldots, v_m\}$ is a finite sequence of vectors in $\mathbb{C}^d$ satisfying $\|v_i\|_2 = \frac{1}{\sqrt{N}}$, $\forall i \in [m]$ and*

$$\sum_{i=1}^m |\langle u, v_i\rangle|^2 = 1$$

for **all** *unit vector u, then the index set $\{1, \ldots, m\}$ can be partitioned into subsets $S_1, \ldots, S_r$ such that*

$$\sum_{i\in S_j} |\langle u, v_i\rangle|^2 \leq 1 - \frac{1}{\sqrt{N}}$$

holds for **all** *unit vector u and all $j = 1, \ldots r$.*

It is not clear whether using $r > 2$ gives us any extra power, therefore we will focus our discussion on the case $r = 2$. This reduces the problem into whether it is possible to find a subset S satisfying $\frac{1}{\sqrt{N}} \leq \sum_{i\in S} |\langle u, v_i\rangle|^2 \leq 1 - \frac{1}{\sqrt{N}}$ for all unit vectors. Note that we require the subset to both have an upper bound and lower bound, which can be thought as a two-sided bound on set S, and it is still open whether a polynomial time algorithm exists. Instead, we consider a simplified version, which can be viewed as a one-sided problem of Kadison-Singer:

Question 6.4. *Does there exist a constant $N \in \mathbb{N}$, such that if $\{v_1, \ldots, v_m\}$ is a finite sequence of vectors in $\mathbb{C}^d$ satisfying $\|v_i\| = \frac{1}{\sqrt{N}}, \forall i \in [m]$, and*

$$\sum_{i=1}^m |\langle u, v_i\rangle|^2 = 1$$

for all unit vector $u \in \mathbb{C}^d$, then there exists a subset $S \subseteq \{1, \dots, m\}$ such that for any $q \in (0,1)$, we have

$$\sum_{i \in S} |\langle u, v_i \rangle|^2 \leq q - \frac{1}{\sqrt{N}}$$

for all unit vector $u \in \mathbb{C}^d$?

In Weaver's discrepancy theory II, 2013 [Wea13], he presented a polynomial algorithm that has the following guarantee:

$$\sum_{i \in S} |\langle u, v_i \rangle|^2 \leq \frac{n}{m} + O(\frac{1}{\sqrt{N}}).$$

for all unit vector u. Here $n := |S|$. We will dedicate our efforts to design a faster algorithmic framework to achieve the same or approximate guarantee as Weaver's result.

6.3 Approximate Greedy Lemma

In this section, we describe and analyze a high level greedy process to construct the set S with the guarantee given in [Wea13]. We generalize his analysis by introducing an approximation factor β, which is particularly valuable when later, we want to use certain approximate data structure to implement the high-level idea. We start with the main lemma of this section.

Lemma 6.5 (approximate greedy lemma)**.** *Let $N \in \mathbb{R}_+$, if $\{v_1, \dots, v_m\}$ is a finite sequence of vectors in $\mathbb{C}^d$ satisfying $\|v_i\|_2 = \frac{1}{\sqrt{N}}, \forall i \in [m]$ and*

$$\sum_{i=1}^{m} |\langle u, v_i \rangle|^2 = 1$$

holds for all unit vector $u \in \mathbb{C}^d$. Then for any $n < m$ and any unit vector u, we can find a set S ($|S| = n$) such that

$$\sum_{i \in S} |\langle u, v_i \rangle|^2 \leq \beta \cdot (\frac{n}{m} + O(\frac{1}{\sqrt{N}})).$$

where $\beta \geq 1$.

Proof. Before proceeding to main body of the proof, we observe that for the choice of N, we know $\mathrm{tr}[v_i v_i^\top] = \|v_i\|_2^2 = \frac{1}{N}$ and $\sum_{i=1}^m v_i v_i^\top = I$, thus we have $m = dN$.

We define the following sequence of numbers

$$a_i = \frac{1}{\sqrt{N}} + (1 + \frac{1}{\sqrt{N} - 1})\frac{i}{m}, \forall i \in \{0, 1, \cdots, n\}.$$

Let S_j to be the set we have at round t, we also define the matrix T_j as

$$T_j := \frac{1}{\beta} \cdot \sum_{i \in S_j} v_i v_i^\top$$

We are going to find a set of indices $i_1, \dots, i_n$ such that the following two things hold

- $\|T_j\| < a_j$,
- $\Phi^{a_0}(T_0) \geq \ldots \geq \Phi^{a_n}(T_n)$.

Assume the above two conditions hold, then we will have

$$\begin{aligned}\left\|\sum_{i\in S_n} v_i v_i^\top\right\| &= \beta \cdot \|T_n\| \\ &< \beta \cdot a_n \\ &= \beta \cdot (\frac{1}{\sqrt{N}} + (1+\frac{1}{\sqrt{N}-1})\frac{n}{m}) \\ &\leq \beta \cdot (\frac{n}{m} + O(\frac{1}{\sqrt{N}})).\end{aligned}$$

Further, since

$$\begin{aligned}\sum_{i\in S_n} \langle u, v_i\rangle^2 &= \sum_{i\in S_n} u^\top (v_i v_i^\top) u \\ &= u^\top (\sum_{i\in S_n} v_i v_i^\top) u \\ &\leq \left\|\sum_{i\in S_n} v_i v_i^\top\right\|.\end{aligned}$$

We conclude that a bound on the spectral norm of βT gives a bound on our desired objective.

Therefore, it suffices to show how to construct S_j that satisfies above two conditions. We will prove via induction.

Base case. For base case, consider $j = 0$, note $a_0 = \frac{1}{\sqrt{N}} > 0$ and $T_0 = 0$, so $\|T_0\| < a_0$. For potential, we compute $\Phi^{a_0}(T_0)$:

$$\Phi^{a_0}(T_0) = \mathrm{tr}[(\frac{1}{\sqrt{N}} I)^{-1}] = d\sqrt{N}.$$

Inductive hypothesis. For inductive hypothesis, we suppose for some $j < n$, we have $\|T_j\| < a_j$ and $\Phi^{a_0}(T_0) \geq \ldots \geq \Phi^{a_j}(T_j)$.

Inductive step. We prove for $j+1$. Suppose $v_1, \ldots v_j$ have been chose and we use $\lambda_1 \leq \cdots \leq \lambda_d$ be the eigenvalue of T_j. Then the eigenvalues of $I - T_j$ are $1-\lambda_1 \geq \cdots \geq 1-\lambda_d$ and the eigenvalues of $(a_{j+1}I - T_j)^{-1}$ are $\frac{1}{a_{j+1}-\lambda_1} \leq \cdots \leq \frac{1}{a_{j+1}-\lambda_d}$. Note that T_j is a complex symmetric matrix, we can express it using its eigen-decomposition: $T_j = Q_j^{-1} D_j Q_j$, where $D_j \in \mathbb{C}^{d\times d}$ is a diagonal matrix, whose i-th entry is λ_i.

Then we have

$$\begin{aligned}\mathrm{tr}[(a_{j+1}I - T_j)^{-1}(I - \beta T_j)] &= \mathrm{tr}[Q_j^{-1}(a_{j+1}I - D_j)^{-1}(I - \beta D_j) Q_j] \\ &= \mathrm{tr}[(a_{j+1}I - D_j)^{-1}(I - \beta D_j)] \\ &= \sum_{l=1}^{d} \frac{1}{a_{j+1}-\lambda_l}(1-\beta\lambda_l)\end{aligned}$$

$$\begin{aligned}
&\le \frac{1}{d}\sum_{l=1}^{d}\frac{1}{a_{j+1}-\lambda_l}\sum_{l=1}^{d}(1-\beta\lambda_l)\\
&= \frac{1}{d}\cdot \mathrm{tr}[(a_{j+1}I-T_j)^{-1}]\cdot \mathrm{tr}[I-\beta T_j]\\
&\le \frac{1}{d}\cdot \mathrm{tr}[(a_{j}I-T_j)^{-1}]\cdot \mathrm{tr}[I-\beta T_j]\\
&= \frac{1}{d}\Phi^{a_j}(T_j)\cdot \mathrm{tr}[I-\beta T_j]\\
&\le \frac{1}{d}\Phi^{a_0}(T_0)\cdot \mathrm{tr}[I-\beta T_j]\\
&= \sqrt{N}\cdot \mathrm{tr}[I-\beta T_j], \qquad (6)
\end{aligned}$$

where the fourth step follows from sorting inequality 4.4, the sixth step follows from $a_{j+1} > a_j$, the eighth step follows from the inductive hypothesis.

Consequently, we have

$$\begin{aligned}
\Phi^{a_j}(T_j)-\Phi^{a_{j+1}}(T_j) &= \mathrm{tr}[(a_jI-T_j)^{-1}-(a_{j+1}I-T_j)^{-1}]\\
&= \mathrm{tr}[Q_j^{-1}(a_jI-D_j)^{-1}Q_j-Q_j^{-1}(a_{j+1}I-D_j)^{-1}Q_j]\\
&= \mathrm{tr}[(a_jI-D_j)^{-1}-(a_{j+1}I-D_j)^{-1}]\\
&= (a_{j+1}-a_j)\mathrm{tr}[(a_jI-D_j)^{-1}(a_{j+1}I-D_j)^{-1}]\\
&= (1+\frac{1}{\sqrt{N}-1})\frac{1}{m}\cdot \mathrm{tr}[(a_jI-T_j)^{-1}(a_{j+1}I-T_j)^{-1}]\\
&\ge (1+\frac{1}{\sqrt{N}-1})\frac{1}{m}\cdot \mathrm{tr}[(a_{j+1}I-T_j)^{-2}] \qquad (7)
\end{aligned}$$

where the forth step follows from Fact 4.3, and the fifth step follows from $a_{j+1}-a_j = \frac{1}{m}(1+\frac{1}{\sqrt{N}-1})$. The last step follows from

$$\frac{1}{(a_{j+1}-\lambda_l)^2} \le \frac{1}{(a_j-\lambda_l)(a_{j+1}-\lambda_l)}.$$

Furthermore, we have

$$\begin{aligned}
\mathrm{tr}[(a_{j+1}I-T_j)^{-2}(I-\beta T_j)] &= \sum_{l=1}^{d}\frac{1}{(a_{j+1}-\lambda_l)^2}(1-\beta\lambda_l)\\
&\le \frac{1}{d}\sum_{l=1}^{d}\frac{1}{(a_{j+1}-\lambda_l)^2}\sum_{l=1}^{d}(1-\beta\lambda_l)\\
&= \frac{1}{d}\mathrm{tr}[(a_{j+1}I-T_j)^{-2}]\cdot \mathrm{tr}[I-\beta T_j]. \qquad (8)
\end{aligned}$$

Combining Eq. (7) and (8), we get

$$\begin{aligned}
\frac{\mathrm{tr}[(a_{j+1}I-T_j)^{-2}(I-\beta T_j)]}{\Phi^{a_j}(T_j)-\Phi^{a_{j+1}}(T_j)} &\le \frac{m}{d}\frac{\sqrt{N}-1}{\sqrt{N}}\cdot \mathrm{tr}[I-\beta T_j]\\
&= N(1-\frac{1}{\sqrt{N}})\cdot \mathrm{tr}[I-\beta T_j], \qquad (9)
\end{aligned}$$

where the last step follows from $m/d = N$.

Denote $\overline{S} = [m] \setminus S$, then we have

$$\begin{aligned}
&\sum_{i \in \overline{S}} \left(\frac{v_i^\top (a_{j+1} I - T_j)^{-2} v_i}{\Phi^a(T_j) - \Phi^{a_{j+1}}(T_j)} + v_i^\top (a_{j+1} I - T_j)^{-1} v_i \right) \\
= &\frac{\operatorname{tr}[(a_{j+1} I - T_j)^{-2} (I - \beta T_j)]}{\Phi^{a_j}(T_j) - \Phi^{a_{j+1}}(T_j)} + \operatorname{tr}[(a_{j+1} I - T_j)^{-1} (I - \beta T_j)] \\
\leq &N(1 - \frac{1}{\sqrt{N}}) \cdot \operatorname{tr}[I - \beta T_j] + \sqrt{N} \cdot \operatorname{tr}[I - \beta T_j] \\
= &N \cdot \operatorname{tr}[I - \beta T_j] \\
= &m - j.
\end{aligned}$$

The first step follows from $\sum_{i \in S'} v_i v_i^\top = I - \beta T_j \in \mathbb{C}^{d \times d}$, the second step follows from Eq. (6) and (9). The last step follows from $\operatorname{tr}[\beta T_j] = j/N$ and $m = Nd$. Thus we conclude there exists an element of $\overline{S}$ satisfying

$$\frac{v_{i^\star}^\top (a_{j+1} I - T_j)^{-2} v_{i^\star}}{\Phi^{a_j}(T_j) - \Phi^{a_{j+1}}(T_j)} + v_{i^\star}^\top (a_{j+1} I - T_j) v_{i^\star} \leq 1 \leq \beta.$$

Thus choosing $S_{j+1} = S_j \cup \{i^\star\} \subseteq [m]$, and using Lemma 6.2, we conclude $\|T_{j+1}\| < a_{j+1}$ and $\Phi^{a_{j+1}}(T_{j+1}) \leq \Phi^{a_j}(T_j)$. □

Remark 6.6. If we choose $\beta = 1$, then the above theorem reduces to the original version proved by Weaver [Wea13], which corresponds to the exact algorithms. In our generalized version, we show that if we scale down each copy of $v_i v_i^\top$ by a factor of β, then the final bound is just worse by a factor of β, compared to the bound obtained by Weaver. This means that at each step of algorithm, we can tolerate for a vector with only approximately small inner products, as long as we know the approximation ratio, we can scale matrix T down and pay back the factor at the final bound. This inspires the use of data structure that outputs approximate solution.

6.4 Algorithm Overviews

In this section, we implement and discuss various exact and approximate algorithms for Lemma 6.5. We first list a table comparing and contrasting various algorithms. Note that the number of iterations for all algorithms are n. The last two rows represent the same algorithms with different approximation guarantees.

Algorithm	**Prep.**	**Cost per Iter.**	**Total Time**	**Comments**
Alg. 1	0	md^ω	$n \cdot md^\omega$	Straightforward Alg. for [Wea13]
Alg. 2	0	$md^{\omega-1}$	$n \cdot md^{\omega-1}$	Smarter Alg. for [Wea13]
Alg. 3	$m^{d^2/2}$	d^ω	$n \cdot d^\omega + m^{d^2/2}$	Our exact algorithm
Alg. 4	$m^{1.5} d^4$	$m^{0.5} d^4$	$n \cdot m^{0.5} d^4 + m^{1.5} d^4$	Ours with high accuracy
Alg. 4	$m^{1.01} d^4$	$m^{0.01} d^4$	$n \cdot m^{0.01} d^4 + m^{1.01} d^4$	Ours with low accuracy

Table 3: Comparison of different algorithms, for simplicity of presentation assume $m \gg d$ and ignore $\widetilde{O}$. All algorithms require n iterations. **Prep.** denote the preprocessing time of data structures. Full version of Table 1.

Algorithm 1 Vanilla greedy algorithm derived from [Wea13], it takes $nm \cdot \mathcal{T}_{\mathrm{mat}}(d,d,d)$ time.

```
1:  procedure VanillaGreedy($\{v_1,\dots,v_m\}, N, n$)    ▷ Theorem 6.7
2:      $T_0 \leftarrow \mathbf{0}_{d\times d}$
3:      $S \leftarrow \emptyset$
4:      for $j = 0 \to n$ do
5:          $a_j = \frac{1}{\sqrt{N}} + (1 + \frac{1}{\sqrt{N}-1})\frac{j}{m}$
6:      end for
7:      for $j = 0 \to n$ do
8:          for $i \in [m] \setminus S$ do
9:              $c_i \leftarrow (\Phi^{a_{j-1}}(T_j) - \Phi^{a_j}(T_j))^{-1} \cdot v_i^\top (a_j I - T_j)^{-2} v_i + v_i^\top (a_j I - T_j)^{-1} v_i$    ▷ $\mathcal{T}_{\mathrm{mat}}(d,d,d)$ time
10:         end for
11:         $i^* = \arg\min_{i\in[m]\setminus S} c_i$
12:         $T_{j+1} \leftarrow T_j + v_{i^*} v_{i^*}^\top$
13:         $S \leftarrow S \cup \{i^*\}$
14:     end for
15:     return $S$
16: end procedure
```

We provide formal correctness and runtime analysis of the four algorithms we presented above. Note due to Lemma 6.5, the correctness boils down to assert that each algorithm can find a vector v_i such that

$$\frac{v_i^\top (a_{j+1} I - T_j)^{-2} v_i}{\Phi^{a_j}(T_j) - \Phi^{a_{j+1}}(T_j)} + v_i^\top (a_{j+1} I - T_j) v_i \leq 1.$$

When using approximate algorithm/data structure, we might not find such a vector with the threshold of 1. Fortunately, according to Lemma 6.5, if we can only find a vector with threshold β, we can obtain a β-approximate solution by scaling down each $v_i v_i^\top$ by a factor of β.

Algorithm 2 Smarter implementation, it takes $n \cdot (\mathcal{T}_{\mathrm{mat}}(d,d,d) + \mathcal{T}_{\mathrm{mat}}(d,d,m) + md)$ time.

```
1:  procedure ImprovedGreedy($\{v_1, \dots, v_m\}, N, n$)        ▷ Theorem 6.8
2:      $T_0 \leftarrow \mathbf{0}_{d \times d}$
3:      $S \leftarrow \emptyset$
4:      for $j = 0 \to n$ do
5:          $a_j = \frac{1}{\sqrt{N}} + (1 + \frac{1}{\sqrt{N}-1})\frac{j}{m}$
6:      end for
7:      for $j = 0 \to n$ do
8:          $M_j \leftarrow (a_j I - T_j)^{-1}$        ▷ $M_j \in \mathbb{R}^{d \times d}$, it takes $\mathcal{T}_{\mathrm{mat}}(d,d,d)$ time
9:          $N_j \leftarrow (a_{j-1} I - T_j)^{-1}$
10:         $Q_j \leftarrow M_j \cdot V$        ▷ $Q_j \in \mathbb{R}^{d \times m}, V \in \mathbb{R}^{d \times m}$, it takes $\mathcal{T}_{\mathrm{mat}}(d,d,m)$ time
11:         for $i \in [m] \setminus S$ do
12:             $c_i \leftarrow (\mathrm{tr}[N_j] - \mathrm{tr}[M_j])^{-1} \cdot (Q_{j,i})^\top Q_{j,i} + v_i^\top Q_{j,i}$        ▷ $O(d)$ time, $Q_{j,i}$ here denotes $i$-th column of $Q_j$
13:         end for
14:         $i^* = \arg\min_{i \in [m] \setminus S} c_i$
15:         $T_{j+1} \leftarrow T_j + v_{i^*} v_{i^*}^\top$
16:         $S \leftarrow S \cup \{i^*\}$
17:     end for
18:     return $S$
19: end procedure
```

Algorithm 3 Our Exact Min-IP data structure implementation.

```
data structure VoronoiDiaragm                                        ▷ Theorem 5.3
    Init($d \in \mathbb{N}$, $m \in \mathbb{N}$, $Y \subset \mathbb{R}^d$)
    Insert($x \in \mathbb{R}^d$)
    Delete($x \in \mathbb{R}^d$)
    Query($x \in \mathbb{R}^d$)
end data structure

procedure Exact($d \in \mathbb{N}$, $m \in \mathbb{N}$, $n \in \mathbb{N}$, $V \subset \mathbb{R}^d$)      ▷ Theorem 6.9
                                                                      ▷ $V := \{v_1, \cdots, v_m\}$
    $T_0 \leftarrow \mathbf{0}_{d\times d}$
    $S \leftarrow \emptyset$
    Construct $V_{\mathrm{vec}} \subset \mathbb{R}^{d^2}$          ▷ $V_{\mathrm{vec}} := \{\mathrm{vec}(v_1 v_1^\top), \cdots, \mathrm{vec}(v_m v_m^\top)\}$
    for $j = 0 \to n$ do
        $a_j = \frac{1}{\sqrt{N}} + (1 + \frac{1}{\sqrt{N}-1})\frac{j}{m}$
    end for
    /*We preprocess $v_1, \cdots v_m$ and build a data-structure   ▷ This takes $O(m\log m + m^{d^2/2})$
    VoronoiDiaragm vd
    vd.Init($d^2, m, V_{\mathrm{vec}}$)
    for $j = 0 \to n$ do
        $M_j \leftarrow (a_j I - T_j)^{-1}$                  ▷ $M_j \in \mathbb{R}^{d\times d}$, it takes $\mathcal{T}_{\mathrm{mat}}(d,d,d)$ time
        $N_j \leftarrow (a_{j-1} I - T_j)^{-1}$              ▷ $N_j \in \mathbb{R}^{d\times d}$, it takes $\mathcal{T}_{\mathrm{mat}}(d,d,d)$ time
        /*We use data-structure to find $i$/                  ▷ This step takes $d^2 \cdot \log m$
        $q \leftarrow \mathrm{vec}((\mathrm{tr}[N_j] - \mathrm{tr}[M_j])^{-1} M_j M_j + M_j))$
        $i^* \leftarrow$ vd.Query($q$)
        $T_{j+1} \leftarrow T_j + v_{i^*} v_{i^*}^\top$
        $S \leftarrow S \cup \{i^*\}$
        vd.Delete($\mathrm{vec}(v_{i^*} v_{i^*}^\top)$)
    end for
    return $S$
end procedure
```

Algorithm 4 Our Approximate Min-IP data structure implementation.

```
1: data structure DS
2:     Init($d \in \mathbb{N}$, $m \in \mathbb{N}$, $Y \subset \mathbb{R}^d$, $c \in (0,1)$, $\tau \in (0,1)$)    ▷ $c$ and $\tau$ are Min-IP parameters.
3:     Insert($x \in \mathbb{R}^d$)
4:     Delete($x \in \mathbb{R}^d$)
5:     Query($x \in \mathbb{R}^d$)
6: end data structure
7: procedure Approximate($d \in \mathbb{N}$, $m \in \mathbb{N}$, $n \in \mathbb{N}$, $V \subset \mathbb{R}^d$, $c \in (0,1)$,$\tau \in (0,1)$)    ▷ Theorem 6.11
8:     $T_0 \leftarrow \mathbf{0}_{d\times d}$
9:     $S \leftarrow \emptyset$
10:    Construct $V$    ▷ $V = [v_1, v_2, \cdots, v_m]$
11:    for $j = 0 \to n$ do
12:        $a_j = \frac{1}{\sqrt{N}} + (1 + \frac{1}{\sqrt{N}-1})\frac{j}{m}$
13:    end for
14:    $Y \leftarrow [v_1 v_1^\top, \cdots, v_m v_m^\top]$
15:    DS ds
16:    ds.Init($d^2$,$m$,$Y$,$c$,$\tau$)
17:    for $j = 0 \to n$ do
18:        $M_j \leftarrow (a_j I - T_j)^{-1}$    ▷ $M_j \in \mathbb{R}^{d\times d}$, it takes $\mathcal{T}_{\mathrm{mat}}(d,d,d)$ time
19:        $N_j \leftarrow (a_{j-1} I - T_j)^{-1}$    ▷ $N_j \in \mathbb{R}^{d\times d}$, it takes $\mathcal{T}_{\mathrm{mat}}(d,d,d)$ time
20:        $q \leftarrow \mathrm{vec}((\mathrm{tr}[N_j] - \mathrm{tr}[M_j])^{-1} M_j M_j + M_j))$
21:        $i^* \leftarrow$ ds.Query($q$)
22:        $T_{j+1} \leftarrow T_j + v_{i^*} v_{i^*}^\top$
23:        $S \leftarrow S \cup \{i^*\}$
24:        ds.Delete($\mathrm{vec}(v_{i^*} v_{i^*}^\top)$)
25:    end for
26:    return $S$
27: end procedure
```

6.5 An $O(nm \cdot \mathcal{T}_{\mathrm{mat}}(d,d,d))$ Implementation

Note that Algorithm 1 is a straightforward implementation of the process derived from the proof of Lemma 6.5.

Theorem 6.7. *Let $N \in \mathbb{N}_+$, if $\{v_1, \ldots, v_m\}$ is a finite sequence of vectors in $\mathbb{C}^d$ satisfying $\|v_i\|_2 = \frac{1}{\sqrt{N}}, \forall i \in [m]$ and $\sum_{i=1}^m |\langle u, v_i\rangle|^2 = 1$ holds for all unit vector $u \in \mathbb{C}^d$. Then for any $n < m$, there exists a deterministic algorithm that takes time $O(n \cdot m\mathcal{T}_{\mathrm{mat}}(d,d,d))$ to find a set S with cardinality n such that*

$$\sum_{i \in S} |\langle u, v_i\rangle|^2 \le \frac{n}{m} + O(\frac{1}{\sqrt{N}}),$$

Proof. The correctness proof is straightforward, since Algorithm 1 implements the greedy process exactly. To analyze the runtime, note the expensive step is to compute quantity c_i at each iteration, where it involves inverting a $d \times d$ matrix, which takes $\mathcal{T}_{\mathrm{mat}}(d,d,d)$ time, and compute the quantity in the form of $v^\top A^{-1} v$, which takes $O(d^2)$ time. Note that at each round, we need to compute c_i for at most m vectors, and there are n rounds. Thus, the total running time is

$$O(nm \cdot \mathcal{T}_{\mathrm{mat}}(d,d,d)).$$

□

6.6 An $n \cdot \mathcal{T}_{\mathrm{mat}}(d,d,m)$ Implementation

There's ample room to optimize the implementation of Algorithm 1, e.g., pre-computing the matrix $(a_j I - T_j)^{-1}$ and $(a_{j-1} I - T_j)^{-1}$ at each iteration and computing c_i in a batched fashion using matrix multiplication. This reduces the cost per iteration from $O(m\mathcal{T}_{\mathrm{mat}}(d,d,d))$ to $\mathcal{T}_{\mathrm{mat}}(d,d,m)$, which is an improvement from $O(md^\omega)$ to $O(md^{\omega-1})$.

Theorem 6.8. *Let $N \in \mathbb{N}_+$, if $\{v_1, \ldots, v_m\}$ is a finite sequence of vectors in $\mathbb{C}^d$ satisfying $\|v_i\|_2 = \frac{1}{\sqrt{N}}, \forall i \in [m]$ and $\sum_{i=1}^m |\langle u, v_i\rangle|^2 = 1$ holds for all unit vector $u \in \mathbb{C}^d$. Then for any $n < m$, there exists a deterministic algorithm that takes time $O(n \cdot \mathcal{T}_{\mathrm{mat}}(d,d,m))$ to find a set S with cardinality n such that*

$$\sum_{i \in S} |\langle u, v_i\rangle|^2 \le \frac{n}{m} + O(\frac{1}{\sqrt{N}}),$$

Proof. Note that by definition, we have $(\mathrm{tr}[N_j] - \mathrm{tr}[M_j])^{-1} = (\Phi^{a_{j-1}}(T_j) - \Phi^{a_j}(T_j))^{-1}$ and

$$\begin{aligned} Q_{j,i}^\top Q_{j,i} &= (M_j \cdot V)_i^\top (M_j \cdot V)_i \\ &= v_i^\top M_j^\top M_j v_i \\ &= v_i^\top (a_j I - T_j)^{-2} v_i, \end{aligned}$$

where the second step follows from each column of V is just v_i, and the last step comes from the fact that M_j is positive definite. Similarly, $v_i^\top Q_{j,i} = v_i^\top (a_j I - T_j)^{-1} v_i$. This proves the correctness of the algorithm. For the runtime, note at each iteration, we need to

- Compute the inverse of $d \times d$ matrix, this takes $\mathcal{T}_{\mathrm{mat}}(d,d,d)$ time;
- Compute the product of one $d \times d$ matrix (M_j) and one $d \times m$ matrix (V), this takes $\mathcal{T}_{\mathrm{mat}}(d,d,m)$ time;

- Compute trace of $d \times d$ matrix and inner product between two length d vectors, this takes $O(d)$ time.

Thus, the cost per iteration is dominated by $\mathcal{T}_{\mathrm{mat}}(d, d, m)$, and the overall running time is

$$n \cdot \mathcal{T}_{\mathrm{mat}}(d, d, m).$$

$\square$

6.7 Exact Data Structure: Curse of Dimensionality and Fast Query

We observe the redundancy in both Algorithm 1 and 2: in order to find the vector with the smallest c_i, we have to compute this value for all candidate vectors. A natural idea is to design a data structure with the following properties:

- After preprocessing, answers the query of finding minimum inner product of the type (3) very efficiently;
- Supporting fast deletion, since we need to remove one vector after adding to set S.

We will make use of Min-IP data structure implemented via exact furthest-neighbor search to realize these requirements.

Theorem 6.9 (Formal version of Theorem 2.1)**.** *Let $N \in \mathbb{N}_+$, if $\{v_1, \dots, v_m\}$ is a finite sequence of vectors in $\mathbb{C}^d$ satisfying $\|v_i\|_2 = \frac{1}{\sqrt{N}}, \forall i \in [m]$ and $\sum_{i=1}^m |\langle u, v_i\rangle|^2 = 1$ holds for all unit vector $u \in \mathbb{C}^d$. Then for any $n < m$, there exists a deterministic algorithm that takes time $\mathcal{T}_{\mathrm{init}} + n \cdot \mathcal{T}_{\mathrm{query}}$ to find a set S with cardinality n such that*

$$\sum_{i \in S} |\langle u, v_i\rangle|^2 \leq \frac{n}{m} + O(\frac{1}{\sqrt{N}}),$$

where $\mathcal{T}_{\mathrm{init}} = O(m \log m + m^{d^2/2})$, $\mathcal{T}_{\mathrm{query}} = O(d^\omega + d^2 \cdot \log m)$. Thus, the total running time is

$$O(m \log m + m^{d^2/2} + n \cdot (d^\omega + d^2 \log m)).$$

Proof. We first illustrate the general idea of Algorithm 3, which makes use of the fact that the LHS of Eq. (2) and Eq. (3). Another equivalence we will establish is between the inner product of two matrices, defined as $\langle A, B\rangle = \mathrm{tr}[A^\top B]$ is equivalent to the inner product after their vectorization. Suppose $A, B \in \mathbb{C}^{d \times d}$, then

$$\begin{aligned}
\mathrm{tr}[A^\top B] &= \sum_{i=1}^d \sum_{j=1}^d A_{i,j} B_{i,j} \\
&= \sum_{i=1}^{d^2} \mathrm{vec}(A)_i \mathrm{vec}(B)_i \\
&= \langle \mathrm{vec}(A), \mathrm{vec}(B)\rangle.
\end{aligned}$$

This is what we implement in Algorithm 3. We first compute all the outer product matrices $v_i v_i^\top$ then vectorize them, and preprocess them into an exact data structure. At each iteration, we compute matrices N_j and M_j as in Algorithm 2, then vectorize the matrix $(\mathrm{tr}[N_j] - \mathrm{tr}[M_j])^{-1} M_j M_j + M_j$

and query the data structure to find the index i^* that gives the smallest inner product. Finally, we need to delete the point $\mathrm{vec}(v_{i^*}v_{i^*}^\top)$ from the data structure. The correctness follows naturally from proceeding discussion.

For the running time, as in Theorem 5.3, preprocessing m data points of d^2 takes time $O(m^{d^2/2} + m \log m)$. At each iteration, we need to compute the following:

- Compute inverse of $d \times d$ matrix, which takes $\mathcal{T}_{\mathrm{mat}}(d, d, d)$ time;
- Compute the vectorization of $d \times d$ matrix, which takes $O(d^2)$ time;
- Query the data structure, which takes $O(d^2 \log m)$ time;
- Remove the point $\mathrm{vec}(v_{i^*}v_{i^*}^\top)$ from the data structure, which takes $O(d^2 \log m)$ time.

Thus, the overall cost per iteration is $\mathcal{T}_{\mathrm{mat}}(d, d, d) + O(d^2 \log m)$, and the total running time is

$$O(m^{d^2/2} + m \log m + n \cdot (\mathcal{T}_{\mathrm{mat}}(d, d, d) + d^2 \log m)).$$

$\square$

Remark 6.10. The exact data structure via Voronoi diagram suffers from the curse of dimensionality due to its exponential dependence on dimension d in preprocessing phase. Though not feasible in general choices of d, such a framework of using data structure to preprocess all outer product matrices into points and use it for fast query is of particular interest. We will develop approximate regime that uses data structure with much faster preprocessing time but can answer query with approximate solution.

6.8 Approximate Data Structure: Fast Preprocessing and Approximate Query

In order to get rid of the curse of dimensionality incurred in exact data structure, we utilize approximate Min-IP data structure implemented via approximate furthest-neighbor search (AFN). Roughly speaking, such data structure enables fast preprocessing time and answers query with approximate guarantee. This incurs a factor on the final bound of S.

Theorem 6.11 (Formal version of Theorem 2.2). *Let $\tau, c, \delta \in (0, 1)$ and $N \in \mathbb{N}_+$, if $\{v_1, \ldots, v_m\}$ is a finite sequence of vectors in $\mathbb{C}^d$ satisfying $\|v_i\|_2 = \frac{1}{\sqrt{N}}, \forall i \in [m]$ and $\sum_{i=1}^m |\langle u, v_i\rangle|^2 = 1$ holds for all unit vector $u \in \mathbb{C}^d$. Then for any $n < m$ and unit vector $u \in \mathbb{C}^d$, there exists a randomized algorithm that takes time $\mathcal{T}$ to find a set S ($|S| = n$) such that with probability at least $1 - \delta$,*

$$\sum_{i \in S} |\langle u, v_i\rangle|^2 \leq \frac{2}{c} \cdot (\frac{n}{m} + O(\frac{1}{\sqrt{N}})).$$

Further, we have

- *If $c \in (\tau, \frac{8\tau}{7+\tau})$, then $\mathcal{T} = O(m^{1.5} d^4 \log^3 m \log(m/\delta) + n\sqrt{m} d^4 \log^3 m \log(m/\delta))$;*
- *If $c \in (\tau, \frac{400\tau}{399+\tau})$, then $\mathcal{T} = O(m^{1.01} d^4 \log^3 m \log(m/\delta) + nm^{0.01} d^4 \log^3 m \log(m/\delta))$.*

Proof. Note that Algorithm 4 is similar to Algorithm 3 except using an approximate data structure. The major difference is the extra parameters τ and c, one can interpret them as tuning parameters for approximation. Unlike exact data structure such as Voronoi diagram, the AFN data structure can only deal with the inner product bounded by $\tau < 1$, so we need to scale our query point by τ.

As promised by the data structure, it will output an index i^* such that Eq. (3) is at most $\frac{\tau}{c}$, this means

$$\langle v_{i^*}v_{i^*}^\top, \tau \cdot \frac{(a_{j+1}I - T)^{-2}}{\Phi^{a_j}(T) - \Phi^{a_{j+1}}(T)} + (a_{j+1}I - T)^{-1}\rangle \leq \frac{\tau}{c}$$
$$\Rightarrow \langle v_{i^*}v_{i^*}^\top, \frac{(a_{j+1}I - T)^{-2}}{\Phi^{a_j}(T) - \Phi^{a_{j+1}}(T)} + (a_{j+1}I - T)^{-1}\rangle \leq \frac{1}{c}.$$

i.e., we obtain an index with $\frac{1}{c}$ approximation guarantee. As we showed in Theorem 6.5, if we proceed with adding c copies of $v_{i^*}v_{i^*}^\top$, we will end up with the following guarantee:

$$\left\| \sum_{i\in S} v_i v_i^\top \right\| \leq \frac{1}{c} \cdot (\frac{n}{m} + O(\frac{1}{\sqrt{N}})).$$

It remains to show we can have a data structure with such guarantee, we shall make use of Theorem 5.6 combined with the transformation illustrated in 5.11, we complete the proof of correctness of the data structure.

Now, we prove the correctness of the running time.

By Corollary 5.8, if we pick $c \in (\tau, \frac{8\tau}{7+\tau})$, then it takes $O(m^{1.5}d^2\log^3 m)$ time to preprocess and the query time is $O(\sqrt{m}d^2\log^3 m)$.

By Corollary 5.10, if we pick $c \in (\tau, \frac{400\tau}{399+\tau})$, then it takes $O(m^{1.01}d^2\log^3 m)$ time to preprocess and the query time is $O(m^{0.01}d^2\log^3 m)$.

Finally, we need to use $O(d^2\log(m/\delta))$ independent Min-IP data structures, and this incurs an additive $\frac{\tau}{c}$ to the guarantee of inner product, which means we need to blow up the runtime involving preprocessing and query of Min-IP data structure by a factor $O(d^2\log(m/\delta))$, and the quality of approximation becomes $\frac{2}{c}$, with a success probability at least $1-\delta$.

This completes the proof. $\square$

Remark 6.12. In this approximate result, we introduce extra parameters τ and c, note that by the range of c, as long as $\tau < 1$, we must have $c < 1$. Therefore, we would like to pick c close to 1. On the other hand, compare the range for two kinds of preprocessing/query time, we note that $\frac{400\tau}{399+\tau} < \frac{8\tau}{7+\tau}$. This means to achieve a better running time, we have to make c smaller and therefore the approximation ratio worse.

7 Experimental Design via Furthest-Neighbor Search

In this section, we consider the rounding up task for experimental design problem posed in [AZLSW20].

- In Section 7.1, we introduce definitions and formally state the problem.
- In Section 7.2, we state some useful facts and tools for later proofs.
- In Section 7.3, we present our algorithm with Min-IP data structure.
- In Section 7.4, we prove an approximate regret lemma, which will provide a lower bound on the eigenvalue.
- In Section 7.5, we state the lemma that justifies the correctness of our algorithm.
- In Section 7.6, we prove the minimum inner product part of swapping algorithm.

- In Section 7.7, we prove the maximum inner product part of swapping algorithm.
- In Section 7.8, we discuss the implication of the swapping lemma and guide our algorithm design.
- In Section 7.9, we prove the main result of this section.

7.1 Definition

Definition 7.1. Let $\Delta_{d\times d}$ be the class of matrices defined as

$$\Delta_{d\times d} := \{A \in \mathbb{R}^{d\times d} : A \succeq 0, \mathrm{tr}[A] = 1\}.$$

Definition 7.2. Let $\psi : \mathbb{R}^{d\times d} \to \mathbb{R}$ be defined as

$$\psi(A) = -2\mathrm{tr}[A^{1/2}],$$

where $A \in \mathbb{R}^{d\times d}$ is a positive semi-definite matrix.

Definition 7.3. We define the Bregman divergence function associated with ψ, $\Delta_\psi : \mathbb{R}^{d\times d}\times\mathbb{R}^{d\times d} \to \mathbb{R}$ as

$$\Delta_\psi(A, B) = \psi(B) - \psi(A) - \langle \nabla\psi(A), B - A\rangle.$$

Definition 7.4. We define the mirror descent matrices $\widetilde{A}_t \in \mathbb{R}^{d\times d}$ and $A_t \in \mathbb{R}^{d\times d}$ as follows:

$$\widetilde{A}_t := \arg\min_{A\succeq 0}\{\Delta_\psi(A_{t-1}, A) + \alpha\langle F_{t-1}, A\rangle\},$$
$$A_t := \arg\min_{A\in\Delta_{d\times d}} \Delta_\psi(\widetilde{A}_t, A).$$

Definition 7.5. We define a sequence of matrices $A_0, A_1, \ldots \in \mathbb{R}^{d\times d}$ as follows:

$$A_0 := (c_0 I + \alpha Z_0)^{-2},$$

where $c_0 \in \mathbb{R}, Z_0 \in \mathbb{R}^{d\times d}$ is symmetric and $A_0 \succ 0$. We also define A_t as

$$A_t := (c_t I + \alpha Z_0 + \alpha\sum_{l=0}^{t-1} F_l)^{-2},$$

where $c_t \in \mathbb{R}$ is the unique constant such that $A_t \succ 0$ and $\mathrm{tr}[A_t] = 1$.

Note we give two alternative definitions of matrix A_t, as shown in Claim 7.8, these two definitions are equivalent.

Finally, we formally define the rounding up problem for experimental design.

Question 7.6. *Let $\pi \in [0,1]^m$ with $\|\pi\|_1 \le n$ and $\sum_{i=1}^m \pi_i x_i x_i^\top = I_d$. Let $\gamma \ge 3$ and $\varepsilon \in (0, \frac{1}{\gamma}]$. Does there exist a subset $S \subset [m]$ with $|S| \le n$ such that*

$$\lambda_{\min}\big(\sum_{i\in S} x_i x_i^\top\big) \ge 1 - \gamma\cdot\varepsilon?$$

7.2 Useful Facts from Previous Work

In this section, we list the facts and tools that will be useful for our proof. For the complete proofs of these facts, we refer readers to [AZLSW20].

Claim 7.7 (Lemma 2.7 in [AZLSW20]). *Let $\Delta_{d\times d}$ be defined as Definition 7.1. Suppose $A_0 = (c_0 I + \alpha Z_0)^{-2} \in \mathbb{R}^{d\times d}$, where $c_o I + \alpha Z_0 \in \mathbb{R}^{d\times d}$ is positive definite, then for any $U \in \Delta_{d\times d}$,*

$$\Delta_\psi(A_0, U) \leq 2\sqrt{d} + \alpha\langle Z_0, U\rangle.$$

Claim 7.8 (Claim 2.9 in [AZLSW20]). *Let $\widetilde{A}_t, A_t \in \mathbb{R}^{d\times d}$ be the matrices defined in Def. 7.4, if*

$$\alpha v_t^\top A_t^{1/2} v_t < 1,$$

then we have

$$\widetilde{A}_t = (A_{t-1}^{-1/2} + \alpha F_{t-1})^{-2}.$$

Claim 7.9 (Claim 2.10 in [AZLSW20]). *Let $\Delta_{d\times d}$ be defined as in Definition 7.1. Suppose $P_t^\top A_t^{1/2} P_t = [b \;\; d; d \;\; c] \in \mathbb{R}^{2\times 2}$, $J = \mathrm{diag}(1, -1)$, and $2\alpha v_t^\top A_t^{1/2} v_t < 1$ for $v_t \in \mathbb{R}^d$ and $A_t \in \Delta_{d\times d}$. Then*

$$\Big(J + P_t^\top A_t^{1/2} P_t\Big)^{-1} = \Big(J + \begin{bmatrix} b & d \\ d & c \end{bmatrix}\Big)^{-1} \succeq \Big(J + \begin{bmatrix} 2b & 0 \\ 0 & 2c \end{bmatrix}\Big)^{-1}.$$

Claim 7.10 (Claim 2.11 of [AZLSW20]). *Suppose $Z \succeq 0$ is a $d \times d$ PSD matrix with $\lambda_{\min}(Z) \leq 1$. Let $\alpha > 0$ be a parameter and $A = (\alpha Z + cI)^{-2} \in \mathbb{R}^{d\times d}$, where $c \in \mathbb{R}$ is the unique real number such that $A \succeq 0$ and $\mathrm{tr}[A] = 1$. Then*

- $\alpha\langle A^{1/2}, Z\rangle \leq d + \alpha\sqrt{d}$,
- $\langle A, Z\rangle \leq \sqrt{d}/\alpha + \lambda_{\min}(Z)$.

7.3 Algorithm

Algorithm 5 Swapping algorithm with Min-IP data structure

```
1: data structure DS
2:     INIT($d \in \mathbb{N}$, $m \in \mathbb{N}$, $Y \subset \mathbb{R}^d$, $c \in (0,1)$, $\tau \in (0,1)$)    ▷ $c$ and $\tau$ are Min-IP parameters.
3:     INSERT($x \in \mathbb{R}^d$)
4:     DELETE($x \in \mathbb{R}^d$)
5:     QUERY($x \in \mathbb{R}^d$)
6: end data structure
7:
8: procedure SWAP($X \in \mathbb{R}^{m \times d}, \pi \in [0,1]^m, \varepsilon \in (0, 1/\gamma], c \in (0,1), \tau \in (0,1)$)    ▷ Theorem 7.16
9:     $\alpha \leftarrow \sqrt{d}\beta/\varepsilon$ and $T \leftarrow n/\varepsilon$
10:    $X \leftarrow X(X^\top \mathrm{diag}(\pi) X)^{-1/2}$    ▷ Whitening
11:    $S_0 \subseteq [m]$ be an arbitrary subset of support $n$
12:    $t \leftarrow 1$
13:    /* Initialize data structure with $S_0$ */
14:    DS ds
15:    $Y \leftarrow \{x_1 x_1^\top, \cdots, x_m x_m^\top\}$
16:    ds.INIT($d^2$,$m$,$Y$,$c$,$\tau$)
17:    while $t \le T$ and $\lambda_{\min}(\sum_{i \in S_{t-1}} x_i x_i^\top) \le 1 - \gamma\varepsilon$ do
18:        Let $c_t$ be the constant s.t. $(c_t I + \alpha \sum_{i \in S_{t-1}} x_i x_i^\top)^{-2} \in \Delta_{d \times d}$    ▷ Binary search
19:        $A_t \leftarrow (c_t I_d + \alpha \sum_{i \in S_{t-1}} x_i x_i^\top)^{-2}$
20:        $q \leftarrow \mathrm{vec}(\frac{A_t}{c(1-\varepsilon)/n} + 2\alpha A_t^{1/2})$
21:        /* Query $q$ */
22:        $i_t \leftarrow$ ds.QUERY($q$)
23:        $j_t \leftarrow \arg\max_{j \in \overline{S}_{t-1}} B^+(x_j)$    ▷ Def. 7.12 with $\frac{1}{c}$ as $\beta$
24:        $S_t \leftarrow S_{t-1} \cup \{j_t\} \setminus \{i_t\}$
25:        $t \leftarrow t + 1$    ▷ Increase the counter
26:        /* Updating data structure by swapping $j_t$ and $i_t$ */
27:        ds.DELETE($x_{i_t} x_{i_t}^\top$)
28:        ds.INSERT($x_{j_t} x_{j_t}^\top$)
29:    end while
30:    return $S_{t-1}$
31: end procedure
```

7.4 Approximate Regret Lemma

In this section, we prove the approximate regret lemma. The key consequence of this lemma is to provide a lower bound of the eigenvalue $\lambda_{\min}(\sum_{i\in S} x_i x_i^\top)$.

Lemma 7.11 (approximate regret lemma). *Let $\beta \ge 1$. Suppose $F_t = u_t u_t^\top - v_t v_t^\top$ for vectors $u_t, v_t \in \mathbb{R}^d$ and $A_0, \dots, A_{T-1} \in \Delta_{d\times d}$ are defined in Def. 7.5 some constant $\alpha > 0$. Then, if $\alpha v_t^\top A_t^{1/2} v_t < \beta/2$ for all t, we have for any $U \in \Delta_{d\times d}$,*

$$-\sum_{t=0}^{T-1}\langle F_t, U\rangle \le \sum_{t=0}^{T-1}\Big(-\frac{\beta u_t^\top A_t u_t}{\beta + 2\alpha u_t^\top A_t^{1/2} u_t} + \frac{\beta v_t^\top A_t v_t}{\beta - 2\alpha v_t^\top A_t^{1/2} v_t}\Big) + \frac{\beta \Delta_\psi(A_0, U)}{\alpha}.$$

Proof. Throughout the proof, we let $\overline{\alpha} := \frac{\alpha}{\beta}$, note that $\overline{\alpha}$ has the property that $\overline{\alpha} v_t^\top A_t^{1/2} v_t < 1/2$, this enables us to use both Claim 7.8 and 7.9. The proof relies on the mirror descent matrices $\widetilde{A}_t$ and A_t we defined Def. 7.4, we need to modify the definition of $\widetilde{A}_t$ with $\overline{\alpha}$ instead of α. Per Claim 7.8, we know that $\widetilde{A}_t = (A_{t-1}^{-1/2} + \overline{\alpha} F_{t-1})^{-2}$, and because of their definitions, we know that $\nabla\psi(\widetilde{A}_t) - \nabla\psi(A_{t-1}) + \overline{\alpha} F_{t-1} = 0$ where the gradient is evaluated at $\widetilde{A}_t$. This means that

$$\begin{aligned}\langle \alpha F_{t-1}, A_{t-1} - U\rangle &= \langle \nabla\psi(A_{t-1}) - \nabla\psi(\widetilde{A}_t), A_{t-1} - U\rangle \\ &= \Delta_\psi(A_{t-1}, U) - \Delta_\psi(\widetilde{A}_t, U) + \Delta_\psi(\widetilde{A}_t, A_{t-1}) \\ &\le \Delta_\psi(\widetilde{A}_{t-1}, U) - \Delta_\psi(\widetilde{A}_t, U) + \Delta_\psi(\widetilde{A}_t, A_{t-1}). \qquad (10)\end{aligned}$$

Above, the second inequality and the last inequality follow from standard inequlities and generalized Pythagorean Theorem of Bregman divergence. Now, consider the quantity $\Delta_\psi(\widetilde{A}_t, A_{t-1})$:

$$\begin{aligned}\Delta_\psi(\widetilde{A}_t, A_{t-1}) &= \psi(A_{t-1}) - \psi(\widetilde{A}_t) - \langle \nabla\psi(\widetilde{A}_t), A_{t-1} - \widetilde{A}_t\rangle \\ &= -2\mathrm{tr}[A_{t-1}^{-1/2}] + 2\mathrm{tr}[\widetilde{A}_t^{1/2}] + \langle \widetilde{A}_t^{-1/2}, A_{t-1} - \widetilde{A}_t\rangle \\ &= \langle \widetilde{A}_t^{-1/2}, A_{t-1}\rangle + \mathrm{tr}[\widetilde{A}_t^{1/2}] - 2\mathrm{tr}[A_{t-1}^{1/2}] \\ &= \langle A_{t-1}^{-1/2} + \overline{\alpha} F_{t-1}, A_{t-1}\rangle + \mathrm{tr}[\widetilde{A}_t^{1/2}] - 2\mathrm{tr}[A_{t-1}^{1/2}] \\ &= \overline{\alpha}\langle F_{t-1}, A_{t-1}\rangle + \mathrm{tr}[\widetilde{A}_t^{1/2}] - \mathrm{tr}[A_{t-1}^{1/2}]. \qquad (11)\end{aligned}$$

Combining Eqs. (10) and (11) and telescoping t from 1 to T yields

$$\begin{aligned}-\overline{\alpha}\sum_{t=0}^{T-1}\langle F_t, U\rangle &\le \Delta_\psi(A_0, U) - \Delta_\psi(\widetilde{A}_T, U) + \sum_{t=0}^{T-1}\mathrm{tr}[\widetilde{A}_{t+1}^{1/2}] - \mathrm{tr}[A_t^{1/2}] \\ &\le \Delta_\psi(A_0, U) + \sum_{t=0}^{T-1}\mathrm{tr}[\widetilde{A}_{t+1}^{1/2}] - \mathrm{tr}[A_t^{1/2}], \qquad (12)\end{aligned}$$

where the second inequality follows from the non-negativitiy of Bregman divergence.

It remains to upper bound $\mathrm{tr}[\widetilde{A}_{t+1}^{1/2}] - \mathrm{tr}[A_t^{1/2}]$.

Set P_t as $\sqrt{\overline{\alpha}}[u_t \;\; v_t] \in \mathbb{R}^{d\times 2}$ and $J = \mathrm{diag}(1, -1) \in \mathbb{R}^{2\times 2}$, we have $\overline{\alpha} F_t = P_t J P_t^\top$. By the definition of $\widetilde{A}_{t+1}^{1/2}$ and the matrix Woodbury formula (Fact. 4.2), we have

$$\mathrm{tr}[\widetilde{A}_{t+1}^{1/2}] = \mathrm{tr}[(A_t^{-1/2} + P_t J P_t^\top)^{-1}] = \mathrm{tr}[A_t^{1/2} - A_t^{1/2} P_t (J + P_t^\top A_t^{1/2} P_t)^{-1} P_t^\top A_t^{1/2}]. \qquad (13)$$

By linearity of trace operator, it suffices to give a spectral lower bound on the 2×2 matrix $(J + P_t^\top A_t^{1/2} P_t)^{-1/2}$. We will use Claim 7.9 as a lower bound:

$$\begin{aligned}
\operatorname{tr}[\widetilde{A}_{t+1}^{1/2}] - \operatorname{tr}[A_t^{1/2}] = & -\operatorname{tr}[-A_t^{1/2} P_t (J + P_t^\top A_t^{1/2} P_t)^{-1} P_t^\top A_t^{1/2}] \\
\leq & -\operatorname{tr}[-A_t^{1/2} P_t (J + \operatorname{diag}(2\overline{\alpha} u_t^\top A_t^{1/2} u_t, 2\overline{\alpha} v_t^\top A_t^{1/2} v_t))^{-1} P_t^\top A_t^{1/2}] \\
= & -\frac{\overline{\alpha} u_t^\top A_t u_t}{1 + 2\overline{\alpha} u_t^\top A_t^{1/2} u_t} + \frac{\overline{\alpha} v_t^\top A_t v_t}{1 - 2\overline{\alpha} v_t^\top A_t^{1/2} v_t}.
\end{aligned} \tag{14}$$

Plugging Eq. (14) into Eq. (12), we arrive at the desired result:

$$-\sum_{t=0}^{T-1} \langle F_t, U \rangle \leq \sum_{t=0}^{T-1} \left(-\frac{\beta u_t^\top A_t u^t}{\beta + 2\alpha u_t^\top A_t^{1/2} u_t} + \frac{\beta v_t^\top A_t v_t}{\beta - 2\alpha v_t^\top A_t^{1/2} v_t}\right) + \frac{\beta}{\alpha} \Delta_\psi(A_0, U).$$

□

7.5 Approximate Swapping Lemma

The goal of this section is to present and prove Lemma 7.13. We start with a helpful definition.

Definition 7.12 (B functions)**.** Let α, β denote two fixed parameters. Let A denote a fixed matrix. We define function $B^+ : \mathbb{R}^d \to \mathbb{R}$ and $B^- : \mathbb{R}^d \to \mathbb{R}$ as follows:

$$B^+(x) = \frac{\langle A, xx^\top \rangle}{\beta + 2\alpha \langle A^{1/2}, xx^\top \rangle},$$
$$B^-(x) = \frac{\langle A, xx^\top \rangle}{\beta - 2\alpha \langle A^{1/2}, xx^\top \rangle}.$$

Lemma 7.13. *Let $\beta \in [1, \gamma - 1)$ and $\varepsilon \in (0, 1/\gamma]$. For every subset $S \subset [m]$ of cardinality n (let $\overline{S}$ denote $[m] \setminus S$), suppose $\lambda_{\min}(\sum_{i \in S} x_i x_i^\top) \leq 1 - \gamma\varepsilon$ and $A = (cI + \alpha \sum_{i \in S} x_i x_i^\top)^{-2}$, where $c \in \mathbb{R}$ is the unique number such that $A \succeq 0$ and $\operatorname{tr}[A] = 1$. For any $\alpha = \sqrt{d}\beta/\varepsilon$ and $n \geq \frac{6}{\gamma - 1 - \beta} d/\varepsilon^2$, we have*

- *Part 1. There exists $i \in S$ such that $2\alpha x_i^\top A x_i < \beta$ and $B^-(x_i) \leq \frac{1-\varepsilon}{\beta n}$,*
- *Part 2. There exists $j \in \overline{S}$ such that $B^+(x_j) \geq \frac{1}{\beta n}$.*

Proof. In this proof, we will extensively use Claim 7.10, therefore, we pre-compute the value $d + \alpha\sqrt{d}$ and $\sqrt{d}/\alpha$ here for references. By the choice of our α, we have

$$d + \alpha\sqrt{d} = (1 + \frac{\beta}{\varepsilon})d \;, \sqrt{d}/\alpha = \frac{\varepsilon}{\beta}. \tag{15}$$

We also define the quantity $\nu := \min_{i \in S, 2\alpha x_i^\top A x_i < \beta} B^-(x_i)$ which will be used throughout our proof.
The proof directly follows from combining Claim 7.14 and Claim 7.15. □

7.6 Approximate Swapping Lemma, Part 1

In this section, we will prove that as long as we enter the main while loop of the algorithm, we can always find an index $i \in S$ such that $B^-(x_i)$ is small.

Claim 7.14 (Part 1 of Lemma 7.13)**.** *There exists $i \in S$ such that $2\alpha x_i^\top A x_i < \beta$ and $B^-(x_i) \le \frac{1-\varepsilon}{\beta n}$.*

Proof. To demonstrate the existence of such an i, it suffices to show that $\min_{i\in S, 2\alpha x_i^\top A x_i < \beta} B^-(x_i) \le \frac{1-\varepsilon}{\beta n}$, we use ν to denote this minimum value. Note that $\nu > 0$, due to the fact $2\alpha x_i^\top A x_i < \beta$ and A is positive definite. To start off, we first show that there always exists an i such that $2\alpha\langle A^{1/2}, x_i x_i^\top\rangle < 1$. Define $Z = \sum_{i\in S} x_i x_i^\top$, and by definition $A = (cI + \alpha\sum_{i\in S} x_i x_i^\top)^{-2} = (\alpha Z + cI)^{-2}$. Assume for the sake of contradiction that such i does not exists. We have

$$\sum_{i\in S} 2\alpha\langle A^{1/2}, x_i x_i^\top\rangle = 2\alpha\langle A^{1/2}, Z\rangle \ge |S| = n. \tag{16}$$

On the other hand, because $Z \succeq 0$ and $\lambda_{\min}(Z) < 1$, invoking Claim 7.10 we get

$$2\alpha\langle A^{1/2}, Z\rangle \le 2d + 2\alpha\sqrt{d},$$

which contradicts Eq. (16) given the choice of α and $n > 4d/\varepsilon$. Thus, there must exist $i \in S$ such that $2\alpha\langle A^{1/2}, x_i x_i^\top\rangle < 1$. Since we set $\beta \ge 1$, this means we can always find an index i such that $2\alpha\langle A^{1/2}, x_i x_i^\top\rangle < \beta$ holds. By the same token, we also have $\sum_{i\in S}(\beta - 2\alpha\langle A^{1/2}, x_i x_i^\top\rangle) \ge 0$. We claim that

$$(\beta - 2\alpha\langle A^{1/2}, x_i x_i^\top\rangle)\nu \le \langle A, x_i x_i^\top\rangle, \text{ for all } i \in S,$$

because if $2\alpha\langle A^{1/2}, x_i x_i^\top\rangle \ge \beta$ the LHS is non-positive while the RHS is always non-negative due to the positive semi-definiteness of A. Subsequently,

$$\begin{aligned}
\nu &\le \frac{\sum_{i\in S}\langle A, x_i x_i^\top\rangle}{\sum_{i\in S}(\beta - 2\alpha\langle A^{1/2}, x_i x_i^\top\rangle)} \\
&\le \frac{\sqrt{d}/\alpha + \lambda_{\min}(\sum_{i\in S} x_i x_i^\top)}{\beta n - 2d - 2\alpha\sqrt{d}} \\
&\le \frac{\varepsilon/\beta + 1 - \gamma\varepsilon}{\beta n(1 - \beta\varepsilon/3)} \\
&\le \frac{1-\varepsilon}{\beta n}
\end{aligned}$$

where the first step holds because the denominator is strictly positive as we have shown; the second step is due to Claim 7.10; the third step has used our choices α and n and our assumption $\lambda_{\min}(\sum_{i\in S} x_i x_i^\top) \le 1 - \gamma\varepsilon$; and the forth step has used $1 - \beta\varepsilon/3 < 1$. We have thus proved that $\nu \le (1-\varepsilon)/(\beta n)$. This proves the existence of the i we want.

□

7.7 Approximate Swapping Lemma, Part 2

In this section, we prove the other key gredient for the swapping to proceed, i.e., there exists an $j \in \overline{S}$ such that $B^+(x_j)$ is large.

Claim 7.15 (Part 2 of Lemma 7.13)**.** *There exists $j \in \overline{S}$ such that $B^+(x_j) \ge \frac{1}{\beta n}$.*

Proof. Define $t = 1/(\beta n)$. To prove Part 2 it suffices to show that

$$\sum_{j \in \overline{S}} \pi_j \langle A, x_j x_j^\top \rangle \geq t \cdot \sum_{j \in \overline{S}} \pi_j (\beta + 2\alpha \langle A^{1/2}, x_j x_j^\top \rangle), \tag{17}$$

because $\pi_j \geq 0$ for all $j \in [m]$. Recall that $\sum_{j=1}^m \pi_j = n$, $\sum_{j=1}^m \pi_j x_j x_j^\top = I_d$. We then have

$$\begin{aligned}
\sum_{j \in \overline{S}} \pi_j (\beta + 2\alpha \langle A^{1/2}, x_j x_j^\top \rangle) &\leq \beta(n - \sum_{j \in S} \pi_j) + 2\alpha \cdot \sum_{j \in \overline{S}} \pi_j \langle A^{1/2}, x_j x_j^\top \rangle \\
&\leq \beta(n - \sum_{j \in S} \pi_j) + 2\alpha \cdot \sum_{j=1}^m \pi_j \langle A^{1/2}, x_j x_j^\top \rangle \\
&= \beta n - \beta \sum_{j \in S} \pi_j + 2\alpha \langle I, A^{1/2} \rangle \\
&= \beta n - \beta \sum_{j \in S} \pi_j + 2\alpha \cdot \mathrm{tr}[A^{1/2}].
\end{aligned}$$

Similarly,

$$\begin{aligned}
\sum_{j \in \overline{S}} \pi_j \langle A, x_j x_j^\top \rangle &= \langle I - \sum_{j \in S} \pi_j x_j x_j^\top, A \rangle \\
&= \mathrm{tr}[A] - \sum_{j \in S} \pi_j \langle A, x_j x_j^\top \rangle
\end{aligned}$$

Subsequently,

$$\begin{aligned}
&\sum_{j \in \overline{S}} \pi_j \langle A, x_j x_j^\top \rangle - t \cdot \sum_{j \in \overline{S}} \pi_j (\beta + 2\alpha \langle A^{1/2}, x_j x_j^\top \rangle) \\
&\geq \mathrm{tr}[A] - \sum_{j \in S} \pi_j \langle A, x_j x_j^\top \rangle - t \cdot \beta \cdot (n - \sum_{j \in S} \pi_j) - 2\alpha t \cdot \mathrm{tr}[A^{1/2}] \\
&\geq 1 - \sum_{j \in S} \pi_j \langle A, x_j x_j^\top \rangle - t \cdot \beta \cdot (n - \sum_{j \in S} \pi_j) - 2\alpha t \sqrt{d} \\
&= 1 - t\beta n - 2t\alpha\sqrt{d} - \sum_{j \in S} \pi_j (\langle A, x_j x_j^\top \rangle - t\beta) \\
&\geq 1 - t\beta n - 2t\alpha\sqrt{d} - \sum_{j \in S} \max\{\langle A, x_j x_j^\top \rangle - t\beta, 0\} \\
&= 1 - t\beta n - 2t\alpha\sqrt{d} - \sum_{j \in S} (\langle A, x_j x_j^\top \rangle - t\beta) - \sum_{j \in S} \max\{(t\beta - \langle A, x_j x_j^\top \rangle), 0\} \\
&\geq 1 - 2t\alpha\sqrt{d} - \sqrt{d}/\alpha - \lambda_{\min}(\sum_{j \in S} x_j x_j^\top) - \sum_{j \in S} \max\{(t\beta - \langle A, x_j x_j^\top \rangle), 0\} \\
&\geq (\gamma - \beta)\varepsilon - \frac{2d}{\varepsilon n} - \sum_{j \in S} \max\{t\beta - \langle A, x_j x_j^\top \rangle, 0\}
\end{aligned} \tag{18}$$

where the second step follows from Fact 4.1 and $\mathrm{tr}[A] = 1$. The forth step follows from $\pi_j \leq 1$ for all j, the second-to-last step follows from we apply $\sum_{j \in S} \langle A, x_j x_j^\top \rangle \leq \sqrt{d}/\alpha + \lambda_{\min}(\sum_{j \in S} x_j x_j^\top)$ which comes from Claim 7.10. The fifth step comes from the fact that $\max\{x, 0\} - \max\{-x, 0\} = x$. Finally, the last step comes from the choices of α, t and $\lambda_{\min}(\sum_{j \in S} x_j x_j^\top) \leq 1 - \gamma\varepsilon$.

Furthermore, because $(\beta - 2\alpha\langle A^{1/2}, x_i x_i^\top \rangle)\nu \le \langle A, x_i x_i^\top \rangle$ for all $i \in S$, using Claim 7.10 we have

$$\sum_{i \in S'} (\beta\nu - \langle A, x_i x_i^\top \rangle) \le \sum_{i \in S'} 2\nu\alpha\langle A^{1/2}, x_i x_i^\top \rangle \le 2\nu(d + \alpha\sqrt{d}),$$

for all $S' \subseteq S$.

Consider $S' = \{i \in S : \beta t - \langle A, x_i x_i^\top \rangle \ge 0\}$. We then have

$$\begin{aligned} \sum_{j \in S'} \max\{\beta t - \langle A, x_j x_j^\top \rangle, 0\} &= \sum_{j \in S'} (\beta t - \langle A, x_j x_j^\top \rangle) \\ &= \beta(t-\nu)|S'| + \sum_{j \in S'} (\beta\nu - \langle A, x_j x_j^\top \rangle) \\ &\le \beta(t-\nu)n + 2\nu(d + \alpha\sqrt{d}) \\ &\le \varepsilon + \frac{4d/\varepsilon}{n} \end{aligned} \tag{19}$$

where the last two inequalities hold because $t - \nu = \varepsilon/(\beta n) \ge 0$, $|S'| \le |S| = n$, $\nu \le 1/(\beta n)$ and the choice of α.

Combining Eqs.(18) and (19) we arrive at

$$\sum_{j \in \overline{S}} \pi_j \{\langle A, x_j x_j^\top \rangle - t(\beta + 2\alpha\langle A^{1/2}, x_j x_j^\top \rangle)\} \ge (\gamma - 1 - \beta)\varepsilon - \frac{6d}{\varepsilon n}.$$

By choice of n, the RHS of the above inequality is non-negative, which finishes the proof of Eq. (17) and thus also the proof of Part 2. □

7.8 Implication of Swapping Lemma

Note that Lemma 7.13 gives rise to a natural swapping algorithm: at each round, we can find an index $i \in S$ with $2\alpha x_i^\top A x_i < \beta$ and $B^-(x_i) \le \frac{1-\varepsilon}{\beta n}$ and an index $j \in \overline{S}$ with $B^+(x_j) \ge \frac{1}{\beta n}$, then swap them. As demonstrated in Alg. 5, the task of finding i is a search for minimum inner product, which can be implemented via our data structures. On the other hand, the search for j is a maximum inner product search, or so-called nearest-neighbor problem. Unlike our Min-IP data structure, most nearest-neighbor search data structures do not support efficient dynamic insertion and deletion, therefore, we instead perform exhaustive search on j. Notice this is tractable since in our regime n is large compared to $n - m$. Hence, the size of $\overline{S}$ is comparably small, exhaustive search is doable.

We also remark that by considering to finding a β-approximation point instead of an point with distance exactly 1, we require the cardinality of S to be larger since $n \propto \frac{d/\varepsilon^2}{\gamma-1-\beta}$. This is an interesting trade-off compared to the approximate result we get in one-sided Kadison-Singer, where the quality of solution becomes worse when β becomes larger. Here, the quality of solution is unaffected while we have more leeway to pack vectors into S. To some extent, this makes the problem easier similar to a worse quality of solution.

7.9 Main Result

In this section, we present the correctness and runtime analysis of Algorithm 5. The correctness follows from the approximate regret and swap lemma, while the runtime comes from the approximate Min-IP data structure.

Theorem 7.16. *Let $\pi \in [0,1]^m$ with $\|\pi\|_1 \leq n$ and $\sum_{i=1}^m \pi_i x_i x_i^\top = I_d$. Let $\gamma \geq 3$ and $\varepsilon \in (0, \frac{1}{\gamma}]$. Then, there exists a subset $S \subset [m]$ with $|S| \leq n$ such that*

$$\lambda_{\min}(\sum_{i \in S} x_i x_i^\top) \geq 1 - \gamma \cdot \varepsilon.$$

Let $\tau, \delta \in (0,1)$ and $c \in (\frac{1}{\gamma-1}, 1)$. If $n \geq \frac{6d/\varepsilon^2}{\gamma-1-2/c}$ and $\alpha = \sqrt{d}/(c\varepsilon)$, then there exists a randomized algorithm with success probability at least $1-\delta$ and running time $O(d^2 \log(n/\delta) \cdot \mathcal{T}_{\text{init}} + \mathcal{T}_{\text{mat}}(m,d,d) + \frac{n}{\varepsilon} \cdot (d^2 \log(n/\delta) \cdot \mathcal{T}_{\text{query}} + \mathcal{T}_{\text{mat}}(d,d,d) \cdot \log d/(c\varepsilon) + (m-n) \cdot d^2))$. Furthermore,

- *If $c \in (\tau, \frac{8\tau}{7+\tau})$, then $\mathcal{T}_{\text{init}} = O(n^{1.5} d^2 \log^3 n)$ and $\mathcal{T}_{\text{query}} = O(\sqrt{n} d^2 \log^3 n)$, so the total running time is*

$$\begin{aligned} &O(\mathcal{T}_{\text{mat}}(m,d,d) + n^{1.5} d^4 \log^3 n \log(n/\delta)) \\ +\ &O(\frac{n}{\varepsilon} \cdot (\mathcal{T}_{\text{mat}}(d,d,d) \log d/(c\varepsilon) + \sqrt{n} d^4 \log^3 n \log(n/\delta) + (m-n) \cdot d^2)); \end{aligned}$$

- *If $c \in (\tau, \frac{400\tau}{399+\tau})$, then then $\mathcal{T}_{\text{init}} = O(n^{1.01} d^2 \log^3 n)$ and $\mathcal{T}_{\text{query}} = O(n^{0.01} d^2 \log^3 n)$, so the total running time is*

$$\begin{aligned} &O(\mathcal{T}_{\text{mat}}(m,d,d) + n^{1.01} d^4 \log^3 n \log(n/\delta)) \\ +\ &O(\frac{n}{\varepsilon} \cdot (\mathcal{T}_{\text{mat}}(d,d,d) \log d/(c\varepsilon) + n^{0.01} d^4 \log^3 n \log(n/\delta) + (m-n) \cdot d^2)). \end{aligned}$$

Proof. We will show Alg. 5 satisfies the properties in the theorem statement. Similar to the proof of Theorem 6.11, we need to scale down the query point by a factor of τ. This means each query will return an index $i \in S_{t-1}$ such that

$$\frac{x_i^\top A_t x_i}{c(1-\varepsilon)/n} + 2\alpha x_i^\top A_t^{1/2} x_i \leq \frac{1}{c},$$

Set $\beta = \frac{1}{c}$, note this is equivalent to find an index i satisfying the first result of Lemma 7.13, which as we have shown, does exist.

This means that at each iteration, we either have

$$\lambda_{\min}(\sum_{i \in S} x_i x_i^\top) \geq 1 - \gamma\varepsilon,$$

which we are done, or we can find i_t and j_t such that

$$B^-(x_{i_t}) - B^+(x_{j_t}) \leq -\frac{\varepsilon}{\beta n}.$$

Combining this fact with Lemma 7.11 and Claim 7.7, we have

$$\begin{aligned} -\langle Z_0 + \sum_{t=0}^{T-1} F_t, U \rangle \leq &\sum_{t=0}^{T-1} \beta(B^-(x_{i_t}) - B^+(x_{j_t})) + \frac{2\beta\sqrt{d}}{\alpha} \\ \leq &\ -T \cdot \frac{\varepsilon}{n} + 2\varepsilon, \end{aligned}$$

Since we can choose U such that

$$-\langle Z_0 + \sum_{t=0}^{T-1} F_t, U \rangle = -\lambda_{\min}(Z_0 + \sum_{t=0}^{T-1} F_t) = -\lambda_{\min}(\sum_{i \in S_T} x_i x_i^\top),$$

this gives a lower bound on the desired eigenvalue we want:

$$\lambda_{\min}(\sum_{i\in S_T} x_i x_i^\top) \geq T \cdot \frac{\varepsilon}{n} - 2\varepsilon.$$

Since $T = \frac{n}{\varepsilon}$, it is lower bounded by $1-2\varepsilon > 1-\gamma\varepsilon$, and we have completed the proof of correctness.

For the running time, we separately consider initialization and cost per iteration. In initialization phase,

- Computing $X(X^\top \mathrm{diag}(\pi)X)^{-1/2}$ takes $O(\mathcal{T}_{\mathrm{mat}}(m,d,d))$ time;
- Initializing data structure with n random points takes time $O(n^{1.5}d^2\log^3 n)$ or $O(n^{1.01}d^2\log^3 n)$ based on the choice of c (Corollary 5.8 and 5.10);

For each iteration, we perform the following:

- Computing eigen-decomposition of $\sum_{i\in S_{t-1}} x_i x_i^\top$ takes $O(d^3)$ time;
- Using binary search to finding c_t takes $O(d^3\log d/(c\varepsilon))$ since the searching range is $O(\alpha+\sqrt{d})$ and each search takes $\mathcal{T}_{\mathrm{mat}}(d,d,d)$ to form the matrix and compute its trace;
- The time of querying data structure is either $O(\sqrt{n}d^2\log^3 n)$ or $O(n^{0.01}d^2\log^3 n)$;
- The brute force search for j takes $O((m-n)\cdot d^2)$ if we pre-compute A_t and $A_t^{1/2}$.

Finally, similar to Theorem 6.11, we need to use $O(d^2\log(n/\delta))$ copies of independent Min-IP data structures to ensure success probability is at least $1-\delta$, and requires n to be larger. This concludes the proof of running time. $\square$

Algorithm	**Preprocessing**	**Cost per Iter.**	**Total Time**
[AZLSW20]	$md^{\omega-1}$	md^2	$md^{\omega-1}+\varepsilon^{-1}nmd^2$
Alg. 5	$md^{\omega-1}+n^{1.5}d^4$	$n^{0.5}d^4$	$md^{\omega-1}+\varepsilon^{-1}n^{1.5}d^4$
Alg. 5	$md^{\omega-1}+n^{1.01}d^4$	$n^{0.01}d^4$	$md^{\omega-1}+\varepsilon^{-1}n^{1.01}d^4$

Table 4: Comparison of different algorithms for experimental design, for simplicity we assume $n \gg m-n$ and ignore $\widetilde{O}$. All algorithms require $\varepsilon^{-1}n$ iterations. Full version of Table 2.

Remark 7.17. Note that our algorithm improves the vanilla algorithm [AZLSW20] in two-folds: 1). we prove that it is not necessary to find the minimum and maximum index as in their vanilla algorithm, it suffices to find an index meets certain threshold, which can be reformulated into a Min-IP problem. Also, if we only use a β-approximation, the total number of iterations of algorithm and the quality of resulting set is unaffected, the only influenced factor is the size of set S can be larger. 2). our algorithm has better running time when n is large compared to $m-n$, e.g., $m-n = m^{o(1)}$ and $n = m - m^{o(1)}$. We improve the query time for searching in large set S while tolerating the brutal force search in small set $\overline{S}$. This also aligns well with our quantization of S under approximation β, i.e., since we can only get a β-approximation solution, the set S that we require our regime to work becomes larger.

Similar to one-sided Kadison-Singer, our algorithm is "imbalanced". It can efficiently deal with the minimum inner product type queries, but falls short for maximum inner product search (through brutal force search). One can imagine using nearest-neighbor search to facilitate the maximum inner product search. On one hand, it is not clear whether data structures such as locality-sensitive hashing support efficient dynamic operations, on the other hand, it is out of the scope of this paper. We leave it as one of the future directions.

Appendix

A Solving Sub-task of CPM via Furthest-Neighbor Search

Cutting plane method (CPM, Jiang, Lee, Song and Wong [JLSW20]) is popular in solving convex programming problem. Suppose there exists a convex set K in a closed box with radius R, the goal is to either find a point in K or prove that K does not contain a ball of radius ε. A standard CPM solver needs to query a *separation oracle* at each round for a hyperplane to separate the point from set K. This new hyperplane will either be added as a new constraint into the constraint matrix A, or help discard redundant constraint from A. In order to measure the importance of constraints, it is instructive to compute or at least approximate the *leverage score* of each constraint, which is the major computation bottleneck for CPM. As shown in [JLSW20], it is a nontrivial task to maintain all the leverage scores under consecutive updates to constraint matrix A and a diagonal matrix W. It is unclear whether maintaining all leverage scores under updates in time $o(m^2)$ per iteration is achievable. This is also a critical step in CPM, since it has to compute a barrier function which uses all the leverage scores. Instead, we focus on a sub-task, which is to find a constraint with smallest leverage score then discard it. As we will show in this section, this sub-task can be reformulated into a data structure design problem that supports efficient query for small leverage score. By capitalizing the approximate Min-IP data structure, we give an efficient implementation of such a data structure. Though this does not solve the big question of maintaining all leverage scores fast, we point out this is a promising direction for further improvements on CPM.

Throughout this section, we consider the scheme where $m \gg n$.

A.1 Data Structure Design for Min Leverage Score

We consider the following task: let $W \in \mathbb{R}^{m\times m}$ be a diagonal matrix with non-negative diagonal entries, and $A \in \mathbb{R}^{m\times n}$. At each timestamp t, W undergoes a rank r_t update, namely,

$$W^{(t+1)} \leftarrow W^{(t)} + \Delta W$$

where $W^{(t)} \in \mathbb{R}^{m\times m}$ denotes matrix W at t-th iteration, and $\mathrm{rank}(\Delta W) = r_t$. For the sake of discussion, we assume $\frac{1}{T}\sum_{t=1}^{T} r_t$ is small. Given a threshold parameter $s > 0$, our task is to find a row of matrix $\sqrt{W}A$ with its leverage score smaller than or equal s. Before formally defining the task, we define the notion of leverage score:

Definition A.1. Let $A \in \mathbb{R}^{m\times n}$ be a full rank matrix, and we use $a_i \in \mathbb{R}^n$ to denote the i-th row of A, we define the statistical leverage score of row a_i as

$$\tau_i(A) := a_i^\top (A^\top A)^{-1} a_i.$$

We formulate the main task as a data structure design problem:

Definition A.2. We consider a dynamic data structure design problem that has the following stages:

- Initialization : The data structure can observe a matrix $A \in \mathbb{R}^{m\times n}$ and a diagonal matrix $W^{(0)} \in \mathbb{R}^{m\times m}$ with non-negative entries. It can also observe a threshold parameter $s > 0$ for leverage score. It can perform preprocessing on A, $W^{(0)}$ and s.

- Update: At iteration t, the data structure receive a new diagonal matrix $\Delta W^{(t)}$ with the guarantee that $\|\Delta W^{(t)}\|_0 = r_t$. The new $W^{(t)}$ should be updated as $W^{(t-1)} + \Delta W^{(t)}$.

- Query: At each iteration (after update $W^{(t)}$), the data structure can be queried to output an index $i \in m$ such that $\tau_i(\sqrt{W^{(t)}}A) \leq s$.

A.2 Naive Implementation

Naively, we can do the following for each iteration t:

- Compute matrix $(A^\top W^{(t)} A)^{-1}$.
- Compute $(\sqrt{W^{(t)}}A)_{i,*}(A^\top W^{(t)} A)^{-1}(\sqrt{W^{(t)}}A)_{i,*}^\top$ for each $i \in [m]$.
- Output i such that $\tau_i(\sqrt{W^{(t)}}A) \leq s$.

We give a rough runtime analysis. Since $W^{(t)}$ is a diagonal matrix, the first step is essentially multiplying an $n \times m$ matrix by an $m \times n$ then invert the $n \times n$ matrix. The first matrix product can be viewed as doing $\frac{m}{n}$ pairs of $n \times n$ matrix multiplication, which takes $O(mn^{\omega-1})$ time, since $m > n$, the time of first step is $O(mn^{\omega-1})$. The second step takes $O(mn^2)$ time.

Note that we haven't exploited the fact that the change ΔW has only r_t entries, this means we can use low rank updates such as matrix Woodbury identity to efficiently update the inverse, or even the projection. Consider the following algorithm, which instead maintains the projection matrix $\sqrt{W^{(t)}}A(A^\top W^{(t)} A)^{-1}A^\top(\sqrt{W^{(t)}})$:

- Initially compute $M^{(0)} = \sqrt{W^{(0)}}A(A^\top W^{(0)} A)^{-1}A^\top(\sqrt{(W^{(0)})})$.
- Suppose we have maintained the projection $M^{(t-1)} = \sqrt{W^{(t-1)}}A(A^\top W^{(t-1)} A)^{-1}A^\top(\sqrt{(W^{(t-1)})})$.
- Let $S = \{i : \Delta W_{i,i}^{(t)} \neq 0\}$, let $M_S \in \mathbb{R}^{n \times r_t}$ be the r_t columns of M from S, and let $M_{S,S}, \Delta W_{S,S}^{(t)} \in \mathbb{R}^{r_t \times r_t}$ be the r_t rows and columns from S.
- $M^{(t)} = M^{(t-1)} - M_S^{(t-1)} \cdot ((\Delta W_{S,S}^{(t)})^{-1} + (M_{S,S}^{(t-1)})^{-1})^{-1} \cdot (M_S^{(t-1)})^\top$.
- Output i such that $M_{i,i}^{(t)} \leq s$.

The most expensive step of initialization is multiplying an $m \times n$ matrix by an $n \times m$ matrix, which takes $\mathcal{T}_{\text{mat}}(m, n, m)$ time. The third step follows directly from apply matrix Woodbury identity (4.2), and it takes $\mathcal{T}_{\text{mat}}(n, r_t, n)$ time. The last step involves reading m diagonal entries of M. This leads to following simple claim.

Claim A.3. *The above naive maintenance algorithm takes $\mathcal{T}_{\text{mat}}(m, n, m)$ for initialization and $O(m + \mathcal{T}_{\text{mat}}(r_t, n, n))$ time per iteration.*

Note that in the regime where $r_t \leq m$ and n is small, the naive implementation has to pay at least $\Omega(m)$ time to read all diagonal entries of $M^{(t)}$. This is essentially the same as computing all the leverage scores and find its minimum. One natural question is whether it's possible to avoid searching through all leverage scores at each round. Following the natural question we asked in Section 1, here we want to consider

Is it possible to beat $O(m)$ time barrier for this small leverage score search problem?

A.3 Min Leverage Score via AFN

We will show that by using furthest-neighbor search data structure, it is possible to accelerate the bottleneck step of querying the smallest leverage score. Before stating the main proposition, we define some runtime metric with respect to our data structure.

Definition A.4. We define $\mathcal{T}_{\mathrm{init}}(m, d, \tau, c)$ and $\mathcal{T}_{\mathrm{query}}(m, d, \tau, c)$ to be the preprocessing and query for approximate Min-IP data structure (5.6) with m points, dimension d, inner product threshold τ and approximate factor c respectively.

Proposition A.5. *Let $\tau, c \in (0, 1)$. There exists a randomized data structure that can solve the design problem in A.2 with the following guarantee:*

- *Initialization takes $\widetilde{O}_n(m + \mathcal{T}_{\mathrm{init}}(m, n^2, \tau, c))$ time;*
- *Update takes $\widetilde{O}_n(\mathcal{T}_{\mathrm{mat}}(r_t, n, n) + r_t \cdot \mathcal{T}_{\mathrm{query}}(m, n^2, \tau, c))$ time;*
- *Query will output an index $i \in [m]$ such that $\pi_i(\sqrt{W^{(t)}}A) \leq \frac{2s}{c}$ in time $\widetilde{O}_n(\mathcal{T}_{\mathrm{query}}(m, n^2, \tau, c))$.*

Moreover,

- *If $c \in (\tau, \frac{8\tau}{7+\tau})$, then*[4]

$$\mathcal{T}_{\mathrm{init}}(m, n^2, \tau, c) = \widetilde{O}_n(m^{1.5})$$
$$\mathcal{T}_{\mathrm{query}}(m, n^2, \tau, c) = \widetilde{O}_n(m^{0.5})$$

  and

- *If $c \in (\tau, \frac{400\tau}{399+\tau})$, then*

$$\mathcal{T}_{\mathrm{init}}(m, n^2, \tau, c) = \widetilde{O}_n(m^{1.01})$$
$$\mathcal{T}_{\mathrm{query}}(m, n^2, \tau, c) = \widetilde{O}_n(m^{0.01})$$

Proof. Note that in each of Lemma A.7, A.8 and A.9, we have proved each of the bullet point. It remains to justify the runtime $\mathcal{T}_{\mathrm{init}}$ and $\mathcal{T}_{\mathrm{query}}$, which follows from the guarantee of approximate Min-IP data structure under different setups of c, see Corollary 5.8 and 5.10. $\square$

Remark A.6. For UPDATE time, note that $\mathcal{T}_{\mathrm{mat}}(r_t, n, n)$ can be upper bounded via $O(nr_t^{\omega-1})$ assume $r_t < n$, therefore, the runtime of UPDATE is dominated by the term $\widetilde{O}_n(r_t \cdot \mathcal{T}_{\mathrm{query}}(m, n^2, \tau, c))$, since it requires at least $O(n^2 \cdot r_t)$ time. When number of iterations T becomes larger, the initialization time can be effectively amortized across all the iterations. Suppose at each iteration we need one call for each of UPDATE and QUERY, then we have reduced the cost per iteration from a naive $O(m + \mathcal{T}_{\mathrm{mat}}(r_t, n, n))$ to $\widetilde{O}_n(r_t \cdot m^{0.5})$ or even $\widetilde{O}_n(r_t \cdot m^{0.01})$. In the regime where r_t is small and $m \gg n$, this gives a huge improvement on cost per iteration. However, in standard CPM setup, we will have $m = O(n)$. We believe our considered setting where constraint matrix A is tall and thin is also interesting. We also remark that our data structure can only output a vector with approximately small leverage score, and the effect of this approximation factor has not yet been factored into the global error analysis of CPM procedure. We leave it as a future direction to pursue.

[4]We use $\widetilde{O}_n$ to hide $\mathrm{poly}(n, \log m)$.

A.4 Initialization

In this section, we prove the procedure INIT of Algorithm 6 satisfies the following guarantee.

Lemma A.7. *The* INIT *procedure of Algorithm 6 takes time* $\widetilde{O}_n(mn^2 + \mathcal{T}_{\mathrm{init}}(m, n^2, \tau, c))$.

Proof. Note that in INIT, we need to compute M, which involves a product between one $n \times m$ matrix and $m \times n$ matrix and inverting an $n \times n$ matrix, since $m \gg n$, this time is dominated by $\mathcal{T}_{\mathrm{mat}}(n, m, n) \leq O(mn^{\omega-1})$. Next, to compute set V, we need to compute outer product of vectors in $\mathbb{R}^n$, for m of them. This takes $O(mn^2)$ time. Finally, initializing data structure with m points, dimension n^2, distance τ and approximate factor c takes $\mathcal{T}_{\mathrm{init}}(m, n^2, \tau, c)$ time. $\square$

A.5 Update

In this section, we give a runtime analysis of procedure UPDATE.

Lemma A.8. *The* UPDATE *procedure of Algorithm 6 takes time* $O(\mathcal{T}_{\mathrm{mat}}(r_t, n, n) + r_t \cdot \mathcal{T}_{\mathrm{query}}(m, n^2, \tau, c))$.

Proof. For UPDATE, note that computing M^{new} takes

- Inverting an $r_t \times r_t$ matrix, which takes $\mathcal{T}_{\mathrm{mat}}(r_t, r_t, r_t)$ time;
- Computing product of $r_t \times n$ and $n \times n$ matrix, which takes $\mathcal{T}_{\mathrm{mat}}(r_t, n, n)$ time (can be upper bounded by $O(n^2 r_t^{\omega-1})$);
- Computing product of $n \times r_t$ and $r_t \times n$ matrix, which takes $\mathcal{T}_{\mathrm{mat}}(n, r_t, n)$ time (can be upper bounded by $O(n^2 r_t^{\omega-1})$).

This means computing M^{new} takes $O(\mathcal{T}_{\mathrm{mat}}(r_t, n, n)) \leq O(n^2 r_t^{\omega-1})$ time. When updating the data structure, notice we only need to modify r_t points, since $\sqrt{W}$ is a diagonal matrix, computing the outer product of each point only takes $O(n^2)$ time and each query uses $\mathcal{T}_{\mathrm{query}}(m, n^2, \tau, c)$ time, so this step overall takes $O(\mathcal{T}_{\mathrm{mat}}(r_t, n, n) + r_t \cdot \mathcal{T}_{\mathrm{query}}(m, n^2, \tau, c))$ time. $\square$

A.6 Query

We present a lemma for QUERY, which gives a $\frac{1}{c}$ approximate solution.

Lemma A.9. *The* QUERY *procedure of Algorithm 6 outputs an index* $i \in [m]$ *such that* $\tau_i(\sqrt{W}^{(t)} A) \leq \frac{s}{c}$ *in time* $\mathcal{T}_{\mathrm{query}}(m, n^2, \tau, c)$.

Proof. The analysis for QUERY is straightforward, forming the query point takes $O(n^2)$ time and querying the point takes $\mathcal{T}_{\mathrm{query}}(m, n^2, \tau, c)$. The guarantee follows from the approximate Min-IP data structure guarantee. $\square$

Algorithm 6 Fast Min-Leverage Score Data Structure

```
data structure DS
    Init($d \in \mathbb{N}$, $m \in \mathbb{N}$, $Y \subset \mathbb{R}^d$, $c \in (0,1)$, $\tau \in (0,1)$)    ▷ $c$ and $\tau$ are Min-IP parameters.
    Insert($x \in \mathbb{R}^d$)
    Delete($x \in \mathbb{R}^d$)
    Query($x \in \mathbb{R}^d$)
end data structure

data structure MinLeverageScore    ▷ Proposition A.5

members
    $M \in \mathbb{R}^{n \times n}$    ▷ Inverse being maintained
    $A \in \mathbb{R}^{n \times m}$
    $W^{\mathrm{old}} \in \mathbb{R}^{m \times m}$    ▷ Old $W$ matrix
    $s \in \mathbb{R}$
    DS ds
end members

procedure Init($A \in \mathbb{R}^{m \times n}, W^{(0)} \in \mathbb{R}^{m \times m}, s \in \mathbb{R}, c \in (0,1), \tau \in (0,1)$)    ▷ Lemma A.7
    $A \leftarrow A$
    $M \leftarrow (A^\top W A)^{-1}$    ▷ Explicitly computing inverse at initialization
    $V \leftarrow \{(\sqrt{W}A)_{i,*}^\top (\sqrt{W}A)_{i,*} : i \in [m]\}$
    ds.Init($n^2, m, V, c, \tau$)
end procedure

procedure Update($W^{(t)} \in \mathbb{R}^{m \times m}$)    ▷ Lemma A.8
    $\Delta W \leftarrow W^{(t)} - W^{\mathrm{old}}$    ▷ $\Delta W \in \mathbb{R}^{m \times m}$, $\|\mathrm{vec}(\Delta W)\|_0 = r_t$
    $S \leftarrow \{i : \Delta W_{i,i} \neq 0\}$
    Let $\Delta W_{S,S} \in \mathbb{R}^{r_t \times r_t}$ be $r_t$ rows and columns of $\Delta W$ from $S$
    Let $A_S \in \mathbb{R}^{n \times r_t}$ be the $r_t$ columns of $A$ from $S$
        ▷ Compute $M^{\mathrm{new}} = (A^\top W^{(t)} A)^{-1}$ via matrix Woodbury identity
    $M^{\mathrm{new}} \leftarrow M - M A_S^\top (\Delta W_{S,S}^{-1} + A_S^\top M A_S)^{-1} A_S M$
    for $i \in S$ do
        ds.Delete($(\sqrt{W^{\mathrm{old}}}A)_{i,*}^\top (\sqrt{W^{\mathrm{old}}}A)_{i,*}$)
        ds.Insert($(\sqrt{W^{(t)}}A)_{i,*}^\top (\sqrt{W^{(t)}}A)_{i,*}$)
    end for
    $W^{\mathrm{old}} \leftarrow W^{(t)}, M \leftarrow M^{\mathrm{new}}$
end procedure

procedure Query    ▷ Lemma A.9
    $q \leftarrow \frac{1}{s\tau}\mathrm{vec}(M)$
    $i \leftarrow$ ds.Query($q$)
    return $i$
end procedure

end data structure
```

References

[ACW16] Josh Alman, Timothy M Chan, and Ryan Williams. Polynomial representations of threshold functions and algorithmic applications. In *2016 IEEE 57th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 467–476. IEEE, 2016.

[ADLS17] Jayadev Acharya, Ilias Diakonikolas, Jerry Li, and Ludwig Schmidt. Sample-optimal density estimation in nearly-linear time. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1278–1289. SIAM, 2017.

[AGSS18] Nima Anari, Shayan Oveis Gharan, Amin Saberi, and Nikhil Srivastava. Approximating the largest root and applications to interlacing families. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1015–1028. SIAM, 2018.

[ALS$^+$18] Alexandr Andoni, Chengyu Lin, Ying Sheng, Peilin Zhong, and Ruiqi Zhong. Subspace embedding and linear regression with orlicz norm. In *International Conference on Machine Learning (ICML)*, pages 224–233. PMLR, 2018.

[AMS92] Pankaj K Agarwal, Jiří Matoušek, and Subhash Suri. Farthest neighbors, maximum spanning trees and related problems in higher dimensions. *Computational Geometry*, 1(4):189–201, 1992.

[And79] Joel Anderson. Extreme points in sets of positive linear maps on B(H). *Journal of Functional Analysis*, 31:195–217, 1979.

[And81] Joel Anderson. A conjecture concerning the pure states of B(H) and a related theorem. In *Topics in modern operator theory*, pages 27–43. Springer, 1981.

[AS10] Pankaj K Agarwal and R Sharathkumar. Streaming algorithms for extent problems in high dimensions. In *Proceedings of the twenty-first annual ACM-SIAM symposium on Discrete algorithms (SODA)*, pages 1481–1489. SIAM, 2010.

[AV95] David S Atkinson and Pravin M Vaidya. A cutting plane algorithm for convex programming that uses analytic centers. *Mathematical Programming*, 69(1-3):1–43, 1995.

[AW02] R. Ahlswede and A. Winter. Strong converse for identification via quantum channels. *IEEE Transactions on Information Theory*, 48(3):569–579, 2002.

[AZLSW20] Zeyuan Allen-Zhu, Yuanzhi Li, Aarti Singh, and Yining Wang. Near-optimal discrete optimization for experimental design: A regret minimization approach. *Mathematical Programming*, pages 1–40, 2020.

[Bes96] Sergei Bespamyatnikh. Dynamic algorithms for approximate neighbor searching. In *CCCG*, pages 252–257, 1996.

[BLSS20] Jan van den Brand, Yin Tat Lee, Aaron Sidford, and Zhao Song. Solving tall dense linear programs in nearly linear time. In *STOC*, 2020.

[Brä18] Petter Brändén. Hyperbolic polynomials and the Kadison-Singer problem. In *arXiv preprint*. https://arxiv.org/pdf/1809.03255, 2018.

[Bra20] 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.

[Bra21] Jan van den Brand. Unifying matrix data structures: Simplifying and speeding up iterative algorithms. In *Symposium on Simplicity in Algorithms (SOSA)*, pages 1–13. SIAM, 2021.

[BSS12] Joshua Batson, Daniel A Spielman, and Nikhil Srivastava. Twice-ramanujan sparsifiers. *SIAM Journal on Computing*, 41(6):1704–1721, 2012.

[BV02] Dimitris Bertsimas and Santosh Vempala. Solving convex programs by random walks. In *Proceedings of the thiry-fourth annual ACM symposium on Theory of computing (STOC)*, pages 109–115. ACM, 2002.

[BWZ16] Christos Boutsidis, David P Woodruff, and Peilin Zhong. Optimal principal component analysis in distributed and streaming models. In *Proceedings of the forty-eighth annual ACM symposium on Theory of Computing (STOC)*, pages 236–249, 2016.

[CLS19] Michael B Cohen, Yin Tat Lee, and Zhao Song. Solving linear programs in the current matrix multiplication time. In *STOC*, 2019.

[Coh16] Michael Cohen. Improved spectral sparsification and Kadison-Singer for sums of higher-rank matrices. In *Banff International Research Station for Mathematical Innovation and Discovery*. https://open.library.ubc.ca/cIRcle/collections/48630/items/1.0340957, 2016.

[CT06] Peter G. Casazza and Janet Crandell Tremain. The Kadison–Singer problem in mathematics and engineering. *Proceedings of the National Academy of Sciences*, 103(7):2032–2039, Feb 2006.

[CW13] Kenneth L. Clarkson and David P. Woodruff. Low rank approximation and regression in input sparsity time. In *Symposium on Theory of Computing Conference (STOC)*, pages 81–90, 2013.

[CW19] Lijie Chen and Ryan Williams. An equivalence class for orthogonal vectors. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 21–40. SIAM, 2019.

[DLY21] 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)*, 2021.

[GIV01] Ashish Goel, Piotr Indyk, and Kasturi R Varadarajan. Reductions among high dimensional proximity problems. In *SODA*, volume 1, pages 769–778. Citeseer, 2001.

[Ind00] Piotr Indyk. Dimensionality reduction techniques for proximity problems. In *Proceedings of the eleventh annual ACM-SIAM symposium on Discrete algorithms (SODA)*, pages 371–378, 2000.

[Ind03] Piotr Indyk. Better algorithms for high-dimensional proximity problems via asymmetric embeddings. In *Proceedings of the fourteenth annual ACM-SIAM symposium on Discrete algorithms (SODA)*, pages 539–545, 2003.

[Jia21] Haotian Jiang. Minimizing convex functions with integral minimizers. In *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 976–985. SIAM, 2021.

[JLSW20] Haotian Jiang, Yin Tat Lee, Zhao Song, and Sam Chiu-wai Wong. An improved cutting plane method for convex optimization, convex-concave games, and its applications. In *Proceedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing (STOC)*, pages 944–953, 2020.

[JSWZ21] 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)*, 2021.

[Kha80] Leonid G Khachiyan. Polynomial algorithms in linear programming. *USSR Computational Mathematics and Mathematical Physics*, 20(1):53–72, 1980.

[KLS20] Rasmus Kyng, Kyle Luh, and Zhao Song. Four deviations suffice for rank 1 matrices. In *Advances in Mathematics*, 2020.

[KS59] Richard V. Kadison and Isadore M. Singer. Extensions of pure states. *American Journal of Mathematics*, 81(2):383–400, 1959.

[KTE88] Leonid G Khachiyan, Sergei Pavlovich Tarasov, and I. I. Erlikh. The method of inscribed ellipsoids. In *Soviet Math. Dokl*, volume 37, pages 226–230, 1988.

[LS15] Yin Tat Lee and He Sun. Constructing linear-sized spectral sparsification in almost-linear time. In *IEEE 56th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 250–269, 2015.

[LS17] Yin Tat Lee and He Sun. An sdp-based algorithm for linear-sized spectral sparsification. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory (STOC)*, pages 678–687, 2017.

[LSW15] Yin Tat Lee, Aaron Sidford, and Sam Chiu-wai Wong. A faster cutting plane method and its implications for combinatorial and convex optimization. In *56th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, 2015.

[LSZ19] Yin Tat Lee, Zhao Song, and Qiuyi Zhang. Solving empirical risk minimization in the current matrix multiplication time. In *COLT*, 2019.

[LW17] Renato Paes Leme and Sam Chiu-wai Wong. Computing walrasian equilibria: Fast algorithms and structural properties. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 632–651. SIAM, 2017.

[MSS15] Adam W. Marcus, Daniel A. Spielman, and Nikhil Srivastava. Interlacing families II: Mixed characteristic polynomials and the Kadison-Singer problem. *Ann. of Math. (2)*, 182(1):327–350, 2015.

[NN89] Yurii Nesterov and Arkadi Nemirovski. Self-concordant functions and polynomial time methods in convex programming. preprint, central economic & mathematical institute, ussr acad. *Sci. Moscow, USSR*, 1989.

[NN13] Jelani Nelson and Huy L Nguyên. OSNAP: Faster numerical linear algebra algorithms via sparser subspace embeddings. In *54th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 117–126. IEEE, 2013.

[PSSS15] Rasmus Pagh, Francesco Silvestri, Johan Sivertsen, and Matthew Skala. Approximate furthest neighbor in high dimensions. In *International Conference on Similarity Search and Applications*, pages 3–14. Springer, 2015.

[Rud96] Mark Rudelson. Random vectors in the isotropic position, 1996.

[Sar06] Tamás Sarlós. Improved approximation algorithms for large matrices via random projections. In *Proceedings of 47th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, 2006.

[Sho77] Naum Z Shor. Cut-off method with space extension in convex programming problems. *Cybernetics and systems analysis*, 13(1):94–96, 1977.

[Sri10] Nikhil Srivastava. *Spectral sparsification and restricted invertibility.* PhD thesis, Yale University, 2010.

[SWZ17] Zhao Song, David P Woodruff, and Peilin Zhong. Low rank approximation with entrywise ℓ_1-norm error. In *Proceedings of the 49th Annual Symposium on the Theory of Computing (STOC)*, 2017.

[SWZ19] Zhao Song, David P Woodruff, and Peilin Zhong. Relative error tensor low rank approximation. In *SODA*, 2019.

[SY21] 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.

[Vai89] Pravin M Vaidya. A new algorithm for minimizing convex functions over convex sets. In *30th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 338–343, 1989.

[VKSdBO00] Marc Van Kreveld, Otfried Schwarzkopf, Mark de Berg, and Mark Overmars. *Computational geometry algorithms and applications.* Springer, 2000.

[Wea04] Nik Weaver. The Kadison–Singer problem in discrepancy theory. *Discrete mathematics*, 278(1-3):227–239, 2004.

[Wea13] Nik Weaver. The Kadison–Singer problem in discrepancy theory, ii. *https://arxiv.org/pdf/1303.2405.pdf*, 2013.

[Woo49] Max A Woodbury. The stability of out-input matrices. *Chicago, IL*, 9, 1949.

[Woo50] Max A Woodbury. Inverting modified matrices. 1950.

[WZD$^+$20] 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 (NeurIPS)*, 2020.

[XZZ18] Chang Xiao, Peilin Zhong, and Changxi Zheng. Bourgan: generative networks with metric embeddings. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems (NeurIPS)*, pages 2275–2286, 2018.

[Yao82] Andrew Chi-Chih Yao. On constructing minimum spanning trees in k-dimensional spaces and related problems. *SIAM Journal on Computing*, 11(4):721–736, 1982.

[YN76] David B Yudin and Arkadi S Nemirovski. Evaluation of the information complexity of mathematical programming problems. *Ekonomika i Matematicheskie Metody*, 12:128–142, 1976.